UNIVERSITÉ DE PARIS VIII
Route de la Tourelle
75571 PARIS CEDEX 12

VLISP

10.3

MANUEL DE REFERENCE
AOÛT 1978

Jérôme CHAILLOUX

Département d'Informatique
Université de PARIS VIII - Vincennes
Route de la Tourelle
75571 Paris CEDEX 12
Tél : 374 12 50 poste 299

V L I S P 1 0 . 3

M A N U E L D E R E F E R E N C E

Jérôme CHAILLOUX

AOUT 1978

Révision 0

TABLE DES MATIERES

CHAPITRE 1

INTRODUCTION

Le système VLISP 10 . 3, implémenté sur ordinateur PDP 10, est une version du système VLISP de l'Université PARIS-8 VINCENNES.

Le langage et le système sont destinés à l'enseignement et à la recherche en programmation expérimentale, en intelligence artificielle, et en informatique musicale.

VLISP 10 est utilisé depuis trois ans déjà dans de nombreux sites en France et à l'etranger.

VLISP 10 est la plus grosse version de VLISP, déjà implémenté sur une grande variété d'ordinateurs, très anciens (CAE 510, CAB 500), plutôt récents (T1600, SOLAR), grands ordinateurs (DEC PDP 10), et microprocesseurs (Zilog Z80, INTEL 8080).

Il comporte, outre l'interprète du langage VLISP, un compilateur, un chargeur-assembleur, un éditeur-indexeur et de nombreux utilitaires de mise-au-point.

De nombreuses augmentations et améliorations nous ont conduit à refondre ce Manuel de Référence, qui livre une description informelle du langage VLISP 10 et de son système, chaque construction décrite étant illustrée par une grande variété d'exemples ponctuels.

Ce manuel n'est pas réellement destiné aux débutants, mais aux utilisateurs ayant déjà une certaine expérience de LISP. Le lecteur ne doit donc pas espérer une introduction progressive et ordonnée aux constructions du langage : les références avant y seront plus souvent la régle que l'exception dans les exemples.

Le manuel décrit en principe tous les aspects courants du système, mais n'engage l'auteur que sur les principes de base. Le manuel sera sujet à des remaniements, au fur et à mesure de l'évolution normale et souhaitée du système et de ses utilisateurs.

Si vous détectez une situation très anomalique ou une erreur manifeste du système, rassemblez le plus grand nombre d'informations sur son état courant, et contactez-moi toute affaire cessante :

Jérôme CHAILLOUX
Département d'Informatique
Université Paris-8 Vincennes
75012 Paris FRANCE
tél : 374-12-50 Poste 299

Je crois à propos de remercier P. GREUSSAY, H. WERTZ, D. GOOSSENS, J.F. PERROT, B. ROBINET, H. HUITRIC, M. NAHAS, G. PAUL, A. CATTENAT, L. AUDOIRE, Y. DEVILLER, G. ENGLERT, A. ARVEILLER, V. DULONG, J.M. HULLOT, K. ZELASKA, D. RONCIN, G. COSLADO ainsi que tous les autres utilisateurs du système qui ont influencé, par leurs excellentes suggestions la conception du système dont ce manuel est le manuel.

Ce manuel a été édité par l'auteur à l'IRCAM sur ordinateur PDP 10, grâce à la bienveillance et l'intérêt de G. BENNETT et J.C. RISSET.

L'édition a été réalisée avec le programme RF de J.L. RICHER, et l'impression du fichier édité a été réalisée sur l'imprimante électrostatique VERSATEC grâce à la compétence de R. BARA.

1.1 Pour Debuter Avec VLISP 10 . 3

Pour jouer VLISP 10 . 3, il suffit apres le LOGIN de jouer sur le terminal :

.R VLISP

Exemple de session simple avec VLISP 10 . 3

```
.r vlisp                              ; Appel du système

                                      ; Réponse normale du système
  ** LOOKUP ERROR (INPUT) : 0         ; qui n'a pas trouvé le fichier
                                      ; d'initialisations sur le disque.
  ---  ALLO ?  ---                    ; Indique que le système
?                                     ; attend des données sur le terminal
?  ()                                 ; NIL est équivalent à ()
=  NIL                                ; dit l'interprète.
; time = 0 ms ;                       ; Cette évaluation a duré peu de temps.
?
?  567                                ; La valeur d'un nombre est ce nombre
=  567                                ; lui-même.
=  ; time = 0 ms ;
?
?
?  (CAR '(A B C))                     ; Appel plus complexe
=  A
=  ; time = 0 ms ;
?
?  (cdr '(a b c))                     ; Les caractères minuscules sont automatiquement
=  (B C)                              ; convertis en majuscules
=  ; time = 0 ms ;
?
?  (DE FOO (A L)                      ;
?     (COND
?        ((NULL L) 0)
?        ..                           ; .. produit irrémédiablement une erreur
*                                     ; de lecture et re-entre dans la boucle
*                                     ; de lecture normale.
  ** READ error : 1
* --- Last Form=  NIL
.REE
?
?  (DE FOO (A L)                      ; Définition de la fonction qui
?     (COND                           ; ramène le nombre d'occurences
?        ((ATOM L) 0)                 ; de l'atome A dans la liste L.
?        ((EQ (CAR L) A) (ADD1 (FOO A (CDR L))))
?        ((FOO A (CDR L))))))))
=  FOO                                ; La fonction est maintenant définie.
=  ; time = 0 ms ;
?
?  (FOO 'DO '(DO DO DO RE MI RE DO MI RE RE DO))  ; Et elle fonctionne.
=  5
=  ; time = 0 ms ;
?
?  (LIBRARY DEBUG)                    ; Chargement des aides à la mise au point.
SYS:DEBUG.VLI loaded.
=  DEBUG
=  ; time = 1500 ms ;
?
?  (TRACE FOO)                        ; Préparation du traçage de FOO.
=  ; time = 20 ms ;
?
?  (FOO 'RE '(DO DO DO RE MI RE DO MI RE RE DO)) ; Appel avec trace de FOO.
```

```
  1 ---> FOO : (RE (DO DO DO RE MI RE DO MI RE RE DO))
  2 ---> FOO : (RE (DO DO RE MI RE DO MI RE RE DO))
  3 ---> FOO : (RE (DO RE MI RE DO MI RE RE DO))
  4 ---> FOO : (RE (RE MI RE DO MI RE RE DO))
  5 ---> FOO : (RE (MI RE DO MI RE RE DO))
  6 ---> FOO : (RE (RE DO MI RE RE DO))
  7 ---> FOO : (RE (DO MI RE RE DO))
  8 ---> FOO : (RE (MI RE RE DO))
  9 ---> FOO : (RE (RE RE DO))
  10 ---> FOO : (RE (RE DO))
  11 ---> FOO : (RE (DO))
  12 ---> FOO : (RE NIL)
  12 <--- FOO = 0
  11 <--- FOO = 0
  10 <--- FOO = 1
  9 <--- FOO = 2
  8 <--- FOO = 2
  7 <--- FOO = 2
  6 <--- FOO = 3
  5 <--- FOO = 3
  4 <--- FOO = 4
  3 <--- FOO = 4
  2 <--- FOO = 4
  1 <--- FOO = 4
= 4
= ; time = 280 ms ;
?
? (UNTRACE FOO)                    ; Enlève le traçage de FOO
= (FOO)
= ; time = 0 ms ;
?
? (STOP)                           ; Retour au moniteur
Bye                                ; pour faire comme ETV.
EXIT
```

•

Toutefois pour profiter au maximum du système, il est préférable de mettre sur sa partition un fichier de nom CONFIG.INI (son utilité est décrite au chapitre 8 Organisation et Utilisation) qui au minimum doit contenir cette invocation :

```
(CONFIGURATION '(SYS (VLISP . INI)))
```

Ce fichier (qui se trouve sur la partition système) contient un certain nombre de fonctions et de macros qui sont très utiles. Un listage de ce fichier est donné en Appendice A. VLISP ignore la page de répertoire que l'éditeur ETV insère au début des fichiers.

En ce cas l'appel du système VLISP devient :

```
.R VLISP

   VLISP 10.3-21 15-Oct-78 00:00:56 (LIS . JER)
   Pour avoir l'etat de la documentation demande .HELP VLISP
SYS:VLISP.INI loaded.
                                   ; Dans ce cas les 3 lignes précédentes
                                   ; sont imprimées et le fichier
  --- ALLO ? ---                   ; SYS:VLISP.INI est chargé.
= PRET                             ; voila c'est prêt.
= ; time = 2280 ms ;
?
? (WHOIS BENNETT)                  ; Ce fichier contient par exemple
  "GB  Gerald Bennett"             ; la fonction WHOIS qui donne le
```

```
   "BKS Bennett Smith"                    ; noms des gens.

   ---  ALLO ?  ---
=  OK
=  ; time = 3360 ms ;
?
?  (WHOIS KAT)                           ; Encore une utilisation de WHOIS.
   "KLK Kate Kentish"
   "KBZ Katarzyna Zelaska"

   ---  ALLO ?  ---
=  OK
=  ; time = 3320 ms ;
?
?  (DE POL (E)                           ; Définition d'une fonction de
?     (IF (ATOM E) E                     ; polonisation préfixe d'une
?        (LIST (CADR E) (POL (CAR E)) (POL (CADDR E)))))
=  POL                                   ; expression 2-aire
=  ; time = 0 ms ;
?
?  (PRETTY POL)                          ; Demande de re-édition de la fonction
SYS:PRETTY.VLI loaded.                   ; Ce fichier est automatiquement chargé.

  (DE POL (E)
      (IF (ATOM E)
          E
          [(CADR E) (POL (CAR E)) (POL (CADDR E))]))

=  NIL
=  ; time = 2980 ms ;
?
?  (POL '(A * B))                        ; Essai de la fonction.
=  (* A B)
=  ; time = 0 ms ;
?
?  (POL '((A + B) * (A - B)))
=  (* (+ A B) (- A B))
=  ; time = 20 ms ;
?
?  (TRACE POL)                    ; Chargement et mise de la trace.
SYS:DEBUG.VLI loaded.
=  (POL)
=  ; time = 1360 ms ;
?
?  (POL '((B ↑ 2) - (4 * (A * C)))))    ; Essai tracé.
   1 ---> POL : (((B ↑ 2) - (4 * (A * C))))
   2 ---> POL : ((B ↑ 2))
   3 ---> POL : (B)
   3 <--- POL = B
   3 ---> POL : (2)
   3 <--- POL = 2
   2 <--- POL = (↑ B 2)
   2 ---> POL : ((4 * (A * C)))
   3 ---> POL : (4)
   3 <--- POL = 4
   3 ---> POL : ((A * C))
   4 ---> POL : (A)
   4 <--- POL = A
   4 ---> POL : (C)
   4 <--- POL = C
   3 <--- POL = (* A C)
   2 <--- POL = (* 4 (* A C))
   1 <--- POL = (- (↑ B 2) (* 4 (* A C)))
```

```
=   (- (↑ B 2) (* 4 (* A C)))
=   ; time = 200 ms ;
?
?   (UNTRACE POLL)                           ; Tentative de suppresion de trace
    ** UNTRACE : C'est quoi ? POLL           ; Message furieux de UNTRACE.
=   (POLL)
=   ; time = 0 ms ;
?
?   (UNTRACE)                                ; VOILA.
=   (POL)
=   ; time = 0 ms ;
?
? ←                      ; Dans SYS:VLISP.INI, le macro-caractère ←
Bye                      ; est défini comme étant un appel de la
EXIT                     ; fonction STOP, d'où l'effet :
.
```



CHAPITRE 2

L ' INTERPRETE VLISP 10

2.1 Les Objets VLISP

L'interprète VLISP 10 . 3 permet de manipuler des objets de 6 types :
- les atomes litéraux
- les atomes numériques (nombres entiers et flottants)
- les chaines de caractères
- les listes
- les tableaux d'objets VLISP
- les tableaux d'objets binaires (zone code).

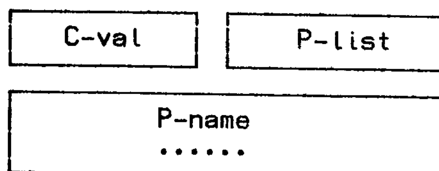
2.1.1 Les Atomes Litéraux -

Ils jouent le rôle d'identificateurs et servent à nommer les variables, les fonctions, les étiquettes ... Ils sont créés implicitement dès leur lecture dans le flux d'entrée ou explicitement par les fonctions IMPLODE ou GENSYM ; nul besoin donc de les déclarer.

Leur nom externe (Print name ou P-name) est une suite de caractères quelconques (contenant au moins un caractère non numérique). A leur création seuls les 13 premiers caractères sont pris en compte. On peut insérer dans un P-name des délimiteurs s'ils sont précédés du caractère / (voir le chapitre 7 sur les entrées/sorties).

Chaque atome possède une valeur propre (sa Cell-Value ou C-val) qui est indéfinie à sa création, ainsi qu'une liste de propriétés (sa Property List ou P-list) qui contient certains aspects particuliers de cet atome par exemple la (ou les) définitions de fonctions qui lui sont associées.

Un atome littéral a approximativement la structure suivante :



En VLISP 10 le CAR d'un atome est sa C-val, le CDR sa P-list. Ces atomes sont stockés dans une zone gérée dynamiquement.

Certains atomes sont déjà connus de l'interprète :
- les constantes littérales (qui contiennent leur propre adresse en C-val) par

exemple : NIL T LAMBDA SUBR FSUBR EXPR FEXPR MACRO MACIN MACOUT QUOTE ...
- les fonctions standard

2.1.2 Les Nombres -

VLISP 10 manipule des nombres entiers (permettant de calculer dans l'intervale : [-2↑35 - 1 , +2↑35]) et des nombres flottants également sur 36 bits.
Le P-name d'un nombre est la représentation de sa valeur dans la base de conversion courante (e.g. 10). La valeur d'un nombre est ce nombre lui-même.
Un nombre n'a ni C-val ni P-list.
Les fonctions sur les nombres ont été regroupées au chapitre 4.

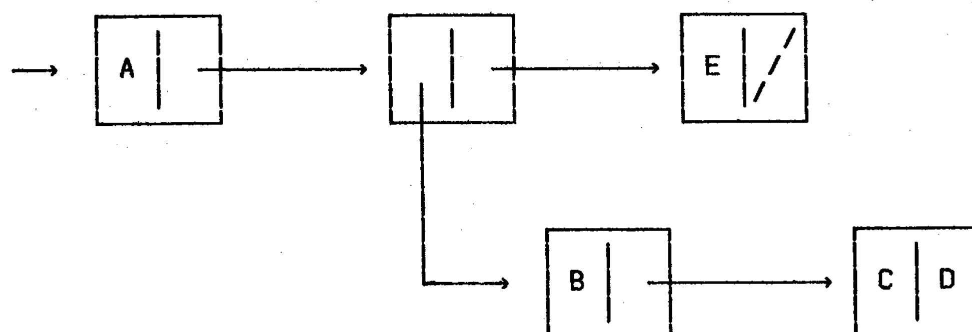
2.1.3 Les Chaines De Caractères -

Les chaines de caractères, considérées en VLISP 10 , comme des atomes, ainsi que leurs fonctions associées sont décrites dans le chapitre 5.

2.1.4 Les Listes -

Les listes sont représentées d'une manière standard : les différents éléments d'une liste utilisent un doublet de pointeurs dont la partie gauche (le CAR) contient l'élément, et la partie droite (le CDR) contient un pointeur sur l'élément suivant ou NIL pour le dernier élément.

Ex : la liste (A (B C . D) E) est stockée en mémoire
sous la forme :



2.1.5 Les Tableaux -

Il est possible d'utiliser des tableaux uni-dimensionnés (des vecteurs donc) d'objets VLISP. Le chapitre 6 leurs est consacré.

2.1.6 Le Code -

Un LAP (LISP Assembly Program) permet d'assembler des instructions en pseudo-assembleur PDP 10 et de charger le code résultant en mémoire, dans une extension de la zone mémoire allouée à l'interprète. Cette zone considérée comme un tableau d'objets binaires est gérée par les fonctions spéciales du LAP (voir le chapitre 11).

2.2 Fonctionnement De Base De L'évaluateur

2.2.1 Evaluation Des Atomes -

La valeur d'un atome littéral est sa C-valeur. L'évaluation d'un atome littéral dont la C-valeur est indéfinie (auquel on a pas encore donné de valeur) provoque lors de son évaluation une erreur dont le libellé est :

** undefined variable <at>

dans lequel le nom de l'atome littéral incriminé est imprimé.

La valeur d'un nombre ou d'une chaîne de caractères est ce nombre ou cette chaîne elle-même.

2.2.2 Evaluation D'une Liste -

L'évaluateur considère toujours une liste comme un appel de fonction. Cette liste s'appelle une forme. Le CAR de cette forme est la fonction, le CDR de la forme les arguments de la fonction.
La valeur d'une forme est la valeur ramenée par l'application de la fonction à ses arguments.

2.3 Les Fonctions

La fonction (le CAR de la forme) peut être un atome littéral dans le cas d'une fonction prédéfinie (standard ou utilisateur), un nombre ou une liste qui est :
- soit une lambda-expression qui est la déclaration explicite d'une fonction
- soit une nouvelle forme dont la valeur est la fonction demandée.

Dans le cas où la fonction est un nombre, il correspond à l'appel implicite de la fonction CNTH. Donc l'évaluation de (<n> <l>) ramène le <n>ième élément de la liste 1er argument <l>.

```
ex :  (3 '(A B C D))     ☞  C
      (8 '(1 2 3 4 5)    ☞  NIL
```

VLISP 10 possède 10 types de fonctions prédéfinies :

Les SUBR et les FSUBR écrites en langage machine et traitées par l'évaluateur normal.

Les EXPR, FEXPR et les MACRO écrites en VLISP et traitées par l'évaluateur normal.

Les fonctions tableaux ARRAY et les fonctions AUTOLOAD qui sont des fonctions spéciales traitées également par l'interprète.

Les MACIN et les macro-caractères écrits en VLISP et traitées par le lecteur

VLISP (la fonction READ)

Les MACOUT écrites en VLISP également et traitées par les fonctions d'impression (PRIN1 et PRINT).

Pour chacun des types que l'on va decrire, on précisera :
- la fonction de l'interprète qui les traitent
- le langage utilisé pour son écriture
- le type des arguments (évalués, non-évalués ...)
- les actions prisent s'il y a trop ou pas assez d'arguments fournis à l'appel

2.3.1 Les SUBR -

Les SUBR sont des fonctions écrites en langage machine, résidentes dans l'interprète dès son lancement (les fonctions standard) ou après compilation (les fonctions compilées). Ces fonctions traitées par l'évaluateur possèdent des arguments évalués. Il existe des SUBR à 0, 1, 2, 3 ou N arguments. Pour les SUBR à nombre fixe d'arguments (0, 1, 2 ou 3) s'il y a trop d'arguments fournis à l'appel, ils sont ignorés sans être évalués. L'interprète n'évalue donc que le nombre d'arguments dont il a besoin. D'autre part s'il manque des arguments ils sont supposés être liés à la valeur NIL.
Il n'est pas possible de définir des fonctions de ce type, sauf à écrire directement en LAP de nouvelles fonctions standard.
Ces fonctions sont très nombreuses dans l'interprète standard, de l'ordre de 300.

```
Ex : (CONS 'A)  = (CONS 'A NIL) ☞  (A)
     (CONS (PRIN1 1) (PRIN1 2) (PRIN1 3)) 1 2 ☞  (1 . 2)
```

2.3.2 Les FSUBR -

Les FSUBR sont des fonctions écrites en langage machine, résidentes dans l'interprète dès son lancement (les fonctions standard) ou après compilation (les fonctions compilées). Ces fonctions traitées par l'évaluateur possèdent des arguments en nombre variable qui ne sont pas évalués. Ces fonctions sont principalement utilisées comme fonctions de contrôle de l'interprète ou comme fonctions de manipulation de noms. Elles sont peu nombreuses.

2.3.3 Les EXPR -

Les EXPR sont des fonctions écrites en VLISP et interprétées par les fonctions standard d'évaluation (EVAL et APPLY). Ces fonctions possèdent un nombre quelconque d'arguments qui sont évalués. S'il y a trop d'arguments fournis à l'appel, ils sont ignorés sans être évalués. S'il n'y en a pas assez, ils sont supposés être liés à la valeur NIL.

(DE <at> <l> <s1> ... <sN>) [FSUBR]

permet de definir de nouvelles EXPR. <at> est le nom de l'atome auquel sera rattachee la lambda-expression :

(LAMBDA <l> <s1> ... <sN>)

DE ramène en valeur le nom de la fonction definie.

2.3.4 Les FEXPR -

Les FEXPR sont des fonctions écrites en VLISP et interprétées par les fonctions standard d'évaluation (EVAL et APPLY). Ces fonctions possèdent un nombre quelconque d'arguments qui ne sont pas évalués.

(DF <at> <l> <s1> ... <sN>) [FSUBR]

> permet de définir de nouvelles FEXPR. <at> est le nom de l'atome auquel sera rattachée la lambda-expression :
>
> (LAMBDA <l> <s1> ... <sN>)
>
> DF ramène en valeur le nom de la fonction définie.

2.4 Lambda-expression Et Liaison Des Variables

En VLISP une fonction peut également être décrite explicitement (ou anonymement) au moyen d'une lambda-expression (lambda-fonction) qui est une liste de la forme :

(LAMBDA <lpf> <s1> ... <sN>)

dans laquelle : <lpf> est la liste des paramètres formels.
<s1> ... <sN> est la liste d'expressions qui seront évaluées en séquence, la valeur ramenée étant celle de <sN>.

Des lambda-expressions de ce type sont automatiquement fabriquees par les fonctions de definition DE, DF et DM.

Les arguments sont liés aux variables d'une manière spécifique à chaque type de fonction.

2.4.1 Liaison Des EXPR -

Si la liste des parametres formels <lpf> est :

- une liste de variables. Dans ce cas la liaison s'effectue élément par élément entre la liste des variables (paramètres formels) et la liste des arguments (paramètres actuels). Si la liste des arguments est plus longue que celle des variables, les arguments restants sont ignorés, SANS être évalués. Si la liste des variables est plus longue que celles des arguments, les variables en trop sont liées à NIL et font office de variables locales.

- une variable. Dans ce cas TOUS les arguments sont évalués puis rassemblés dans un liste de valeurs qui est liée à cette variable.

- une liste pointée de variables qui est une variante du cas précédent (voir le dernier exemple).

ex: de liaison des EXPR

```
EXPR  ((LAMBDA (X Y Z) (LIST X Y Z))
           (PRIN1 1) (PRIN1 2) (PRIN1 3) (PRIN1 4))
      1 2 3   et  ☞  (1 2 3)
```

```
EXPR   ((LAMBDA (X Y Z) (LIST X Y Z))
            (PRIN1 1))
       1  et  ☞  (1 NIL NIL)

EXPR   ((LAMBDA X X) (PRIN1 1) (PRIN1 2) (PRIN1 3))
       1 2 3  et  ☞  (1 2 3)

EXPR   ((LAMBDA (X Y . Z) (LIST X Y Z))
            (PRIN1 1) (PRIN1 2) (PRIN1 3) (PRIN1 4))
       1 2 3 4  et  ☞  (1 2 (3 4))
```

2.4.2 Liaison Des FEXPR -

Pour ce type de fonction aucun argument n'est évalué. Tous les arguments sont rassemblés en une liste qui est liée à la première variable du paramètre formel (qui doit être obligatoirement une liste). Toutes les autres variables de cette liste font office de variables locales et sont initialisées à NIL.

ex : de liaison des FEXPR

```
[illegible] (X) X) (PRINT (CAR '(A B))))
         ☞  ((PRINT (CAR '(A B))))

FEXPR   ((LAMBDA (X Y . Z) (LIST X Y Z))
             (PRIN1 1) (PRIN1 2) (PRIN1 3) (PRIN1 4))
         ☞  (((PRIN1 1) (PRIN1 2) (PRIN1 3) (PRIN1 4)) NIL NIL)
```

2.4.3 Liaison Des MACIN Et Des MACOUT -

Cette liaison est identique aux EXPR mais les arguments (le CDR de ce qui a été lu ou ce qui devait être imprimé) ne sont pas évalués.

2.5 Les MACRO

L'interprète reconnait un nouveau type de fonctions, les MACRO. Une macro est un atome qui possède sur sa P-liste une lambda-expression sous l'indicateur MACRO. A l'évaluation d'une forme dont le CAR est une macro, EVAL évaluera d'abord la fonction associée à cette macro avec toute la forme comme argument, puis re-évaluera la valeur retournée de cette première évaluation. L'évaluation d'une macro se fait donc en deux temps.

C'est l'appel de la macro tout entier qui est passé en argument, il est donc possible de modifier physiquement cet appel à la première évaluation de la macro (voir les exemples). Il existe une fonction qui permet de définir aisement des MACRO à la manière des fonctions DE DF DMC DMI ou DMO.

Ces MACRO ne doivent pas être confondues avec les macro-fonctions d'entrée et de sortie (MACIN MACOUT) qui ne sont activées que pendant des lectures ou des écritures.

(DM <at> <l> <s1> ... <sN>) [FSUBR]

définit une fonction de type macro. Cette fonction a pour nom <at>, sa liste d'arguments est <l> et son corps de fonction est <s1> ... <sN>. DM ramène le nom de la fonction en valeur, et peut être définie comme suit :

```
(DF DM (L) (PUT (CAR L) (CONS 'LAMBDA (CDR L)) 'MACRO))
```

exemples d'utilisation des MACRO :

```
; liaison des arguments d'une MACRO ;

? (DM MAC (L)
?     (PRINT 'L '= L)      ; cette MACRO édite son argument et   ;
?     '(ADD1 3))           ; ramène la forme (ADD1 3) qui sera   ;
= MAC                        ; reévaluée. ;

? (MAC 'X)
  L = (MAC 'X)
= 4

; définition de la fonction MCONS sous forme de MACRO ;

? (DM MCONS (L)
?   (IF (NULL (CDDR L))
?       (CADR L)
?       ['CONS (CADR L) (CONS 'MCONS (CDDR L))])))
= MCONS

? (MCONS 'A 'B 'C)
= (A B . C)

; exemple de modification physique du corps des
  fonctions contenant des MACRO ;

? (DM ZEROP (L)
?   (RPLACB L ['EQ (CADR L) 0]))
= ZEROP

? (DE FACT (N)
?   (IF (ZEROP N) 1 (TIMES N (FACT (SUB1 N)))))
= FACT

? (FACT 5)
= 72

? (CDR 'FACT)
  (EXPR (LAMBDA (N)
    (IF (EQ N 0) 1 (TIMES N (FACT (SUB1 N))))))
```

2.6 Le Mode AUTOLOAD

Certaines fonctions utilisées fréquemment, comme les fonctions d'éditions ou de corrections, peuvent être chargées automatiquement à leur premier appel. Ces fonctions doivent se trouver dans un fichier et peuvent être de n'importe quel type (EXPR FEXPR MACRO ARRAY SUBR compilée FSUBR compilée ...). Ce fichier sera lu par la fonction LIBRARY, dans un des répertoires spécifiés par la fonction PATHLIBRARY.

Si la fonction ne se trouve pas dans le fichier indique, une erreur apparait dont le libellé est :
    ** undefined function
La fonction AUTOLOAD permet de déclarer des fonctions de ce type.

(AUTOLOAD <file> <at1> <at2> ... <atN>) [FSUBR]

met l'indicateur AUTOLOAD sur les différents atomes at1 ... atN. Ces fonctions devront se trouver dans le fichier de nom <file.VLI>. Au 1er appel de l'une de ces fonctions, le fichier <file> sera chargé en entier.

ex : (AUTOLOAD PHENAR PHENARETE PHENARETEFILE)
déclare les deux fonctions PHENARETE et PHENARETEFILE de type AUTOLOAD, elles doivent se trouver dans le fichier PHENAR.VLI .

2.7 L'évaluateur

L'évaluateur VLISP 10 . 3 (i.e. les fonctions interprète EVAL et APPLY) ont subi de nombreuses transformations qui les rendent à la fois plus puissantes et plus rapides.

Un mode de lancement rapide pour les fonctions de type SUBR ou FSUBR est utilisé, si ces fonctions n'ont jamais été redéfinies.

L'évaluateur minimise le nombre de CONS internes, en utilisant le moins possible la fonction EVLIS (et ce en particulier pour evaluer les arguments des lambda-expressions).

Une particularité maîtresse de VLISP, quelque soit l'implémentation, est d'interpréter itérativement les fausses récursivités (1). Itérativement est ici défini en termes de ressources : un appel de fonction est itératif s'il ne demande pas plus de ressources que celles accordées à l'entrée de la fonction. Les ressources en questions sont les tailles de piles, ainsi que le nombre de doublets.

C'est ainsi que dans :

```
(DE LOOPEVAL (IT)
    (LOOPEVAL (PRINT (EVAL (READ)))))
```

la suite des appels internes de LOOPEVAL ne provoquera PAS un débordement de pile, et bouclera, comme il se doit pour une boucle-système, indéfiniment.

Cette propriété se conserve, quelque soit le niveau d'imbrication des appels dits itératifs dans les structures de contrôle mises en jeu dans le corps de fonction.

Le système traîte également avec succès, les cas de CO-POST-RECURSION comme dans cette fonction qui imprime des permutations dans l'ordre lexicographique.

```
(DE PERMS (N) (F 0 NIL))
```

---

(1) voir "Contribution à la définition interprétative et à l'implémentation des lambda-langages" par Patrick GREUSSAY, Laboratoire Informatique Théorique et Programmation, Novembre 1977, No 78-2, 2 place Jussieu, 75221 Paris Cedex 05, FRANCE.

```
(DE F (NIV E)
   (IF (LT NIV N)
      (H (ADD1 NIV) (CONS 1 E))
      (PRINT (REVERSE E))
      (G NIV (CAR E) (CDR E))))

(DE G (NIV X E)
   (IF (EQ X N)
      (IF E
         (G (SUB1 NIV) (CAR E) (CDR E))
         'FINI)
      (H NIV (CONS (ADD1 X) E))))

(DE H (NIV E)
   (IF (MEMQ (CAR E) (CDR E))
      (G NIV (CAR E) (CDR E))
      (F NIV E)))

(PERMS 3) ☞
  (1 2 3)
  (1 3 2)
  (2 1 3)
  (2 3 1)
  (3 1 2)
  [illegible]
= FINI
= ; time = 60 ms ;
```

Dans ce cas seuls les premiers appels de F,G et H consommeront de la pile, tous les autres n'en consommeront pas.

L'utilisation systématique de cette propriété induit un style très souple et très naturel de programmation récursive, bien éloigné des horribles structures de contrôle dites plates.

2.8 VLISP 10 En VLISP 10

```
(DE TOP-LEVEL ()
   ; boucle principale ;
   (WHILE T
      (PRINT "Toplevel")
      (SETQ IT (PRINT (ESCAPE ERREUR (EVAL (READ))))))

(DE EVAL (forme)
   ; evaluation d'une forme quelconque ;
   (COND
      ((OR (NUMBP forme) (STRINGP forme)) forme)
      ((LITATOM forme)
         (IF (EQ (CAR forme) 'UNDEF)
            (ERREUR ["Atome indefini : " forme])
            (CAR forme)))
      (T (EVALF (CAR forme) (CDR forme)))))

(DE EVALF (fnt larg)
   ; evaluation d'une fonction ;
   (COND
      ((LITATOM fnt)
         (SELECTQ (TYPEFN fnt)
            (SUBR (SUBR fnt (EVLIS larg)))
            (FSUBR (FSUBR fnt larg))
            (ARRAY (ARRAY fnt (EVAL (CAR larg))))
            (AUTOLOAD (LIBRARY (GET fnt AUTOLOAD)) (EVALF fnt larg))
            (EXPR (EVALF (GET fnt EXPR) larg))
            (FEXPR (APPLY (GET fnt FEXPR) larg))
            (MACRO (EVAL (EVALF (GET fnt MACRO) forme)))
            ((IF (NEQ fnt (CAR fnt))
               (EVALF (CAR fnt) larg)
               (ERREUR ["Fonction indefinie : " fnt])))))
      ((NUMBP fnt) (CNTH fnt (EVAL (CAR larg))))
      ((STRINGP fnt) (ERREUR ["Fonction indefinie : " fnt]))
      (T (IF (EQ (CAR fnt) LAMBDA)
            (APPLY fnt (EVLIS larg))
            (EVALF (EVAL fnt) larg)))))

(DE EVLIS (l)
   ; construit la liste des valeurs des evaluations
     de tous les elements de l ;
   (MAPCAR l 'EVAL))

(DE APPLY (f la ;; lvar tok)
   ; applique la fonction f (lambda-expression)
     a la liste d'arguments la ;
   (SETQ lvar (CADR f))
   (PUSH '**Marker**)
   (WHILE (LISTP lvar)
      (PUSH (CAR lvar) (CAAR lvar))
      (SET (NEXTL lvar) (NEXTL la)))
   (AND lvar
      (PROGN (PUSH lvar (CAR lvar)) (SET lvar la)))
   (PROG1
      (EPROGN (CDDR f))
      (UNTIL (EQ (SETQ tok (POP)) '**Marker**)
         (SET (POP) tok))))

(DE ERROR.UBV (atome) (CAR atome))

(PRINT "Pour lancer l'interprete : (TOP-LEVEL)")
```

CHAPITRE 3

LES FONCTIONS STANDARD

Toutes les fonctions qui vont être décrites dans ce chapitre sont résidentes dans le système actuel.
Pour chacune d'elles on donnera son type (SUBR ou FSUBR) ainsi que le nombre standard d'arguments, et pour chaque argument le type souhaité ces types étant notés :

<s> pour une S-expression quelconque
<l> pour une liste
<a> pour un atome quelconque (atome littéral, nombre ou chaîne)
<at> pour un atome littéral
<c> pour un caractère (i,e, un atome dont le P-length est égal à 1)
<fn> pour une fonction (nom d'atome ou lambda-expression explicite)

Dans la mesure du possible, les SUBR seront décrites en VLISP.

3.1 Les Fonctions Interprète

(EVAL <s>) [SUBR à 1 argument]

C'est la fonction principale de l'interprète. EVAL ramène la valeur de l'évaluation de l'argument <s> (voir la description de cette fonction en VLISP dans le chapitre précédent).

```
ex : (EVAL '(ADD1 55))                ☞  56
     (EVAL '(2 '(A B C))              ☞  B
     (EVAL '((CAR '(CDR)) '(A B C)))  ☞  (B C)
```

(APPLY <fn> <l>) [SUBR à 2 arguments]

ramène la valeur de l'application de la fonction <fn> à la liste d'arguments <l>. La fonction <fn> peut être de n'importe quel type (SUBR FSUBR EXPR FEXPR MACRO ARRAY AUTOLOAD ...)

```
ex : (APPLY 'PLUS (LIST 5 (ADD1 8) (REM 10 3)))  ☞  15
     (APPLY (ADD1 2) '((A B C D E)))             ☞  C
```

(APPLYN <fn> <s1> ... <sN>) [SUBR à N arguments]

est une autre forme de la fonction APPLY.
L'appel (APPLYN <fn> <s1> ... <sN>) est équivalent à l'appel (APPLY <fn> (LIST <s1> ... <sN>)).

```
ex : (APPLYN 'CONS 'A 'B)  ☞  (A . B)
```

(SUBR <fn> <l>) [SUBR à 2 arguments]

applique la définition de type SUBR de la fonction <fn> à la liste d'arguments <l> de la même manière que le ferait la fonction APPLY. Toutefois si <fn> ne possède pas de définition de type SUBR, une erreur apparait dont le libellé est :

** undefined function (SUBR) : <fn>

dans lequel le nom de la fonction incriminée <fn> est imprimé.
Cette fonction permet donc de lancer la définition standard d'une fonction standard de type SUBR qui a été redéfinie par l'utilisateur.

```
ex : (SUBR 'ADD1 '(3))   ☞  4

     (DE CAR (X)
        (PRINT "J'entre dans CAR avec :" X)
        (PRINT "J'en sors avec :" (SUBR 'CAR [X]))
     ☞ CAR

     (CAR '(A B C))
     "J'entre dans CAR avec :" (A B C)
     "J'en sors avec :" A
     ☞ A
```

(FSUBR <fn> <l>) [SUBR à 2 arguments]

applique la définition de type FSUBR de la fonction <fn> à la liste d'arguments <l> de la même manière que le ferait la fonction APPLY. Toutefois si <fn> ne possède pas de définition de type FSUBR, une erreur apparait dont le libellé est :

** undefined function (FSUBR) : <fn>

dans lequel le nom de la fonction incriminée <fn> est imprimé.
Cette fonction permet donc de lancer la définition standard d'une fonction standard de type FSUBR qui a été redéfinie par l'utilisateur.

```
ex : (DF INCR (L)
        (IF (MEMQ (CAR L) '(X Y Z))
           (PRINT "J'incrémente :" (CAR L) '= (FSUBR 'INCR L))
           (FSUBR 'INCR L)))
     ☞  INCR

     (SETQQ A 5 X 3)  ☞  3

     (INCR A)  ☞  6

     (INCR X)    "J'incrémente :" X = 4  ☞  4
```

(EVLIS <l>) [SUBR à 1 argument]

ramène une liste composée des valeurs des évaluations de tous les éléments de la liste <l>.

Cette fonction peut être décrite en VLISP :

```
     (DE EVLIS (l)
        (IF (NULL l)
            ()
            (CONS (EVAL (CAR l)) (EVLIS (CDR l)))))

ex : (SETQ L '((ADD1 5) (ADD1 7) (ADD1 9)))
      ☞   ((ADD1 5) (ADD1 7) (ADD1 9))
     (EVLIS L)   ☞   (6 8 10)
```

(EPROGN <l>) [SUBR à 1 argument]

évalue séquentiellement tous les éléments de la liste <l>. EPROGN ramène en valeur la valeur de la dernière évaluation (i.e. celle du dernier élément de <l>).

Cette fonction peut être décrite en VLISP :

```
(DE EPROGN (l)
   (IF (NULL (CDR l))
       (EVAL (CAR l))
       (EVAL (CAR l))
       (EPROGN (CDR  l))))
```

```
ex : (SETQ L '((PRIN1 1)(PRIN1 2)(PRIN1 3)))
      ☞    ((PRIN1 1)(PRIN1 2)(PRIN1 3))
     (EPROGN L)  1 2 3  ☞  3
```

(PROG1 <s1> ... <sN>) [FSUBR]

évalue séquentiellement les différentes expressions <s1> ... <sN>. PROG1 ramène la valeur de la première évaluation (i.e. celle de <s1>).

```
ex : (PROG1 (PRIN1 1)(PRIN1 2)(PRIN1 3))  1 2 3  ☞  1
```

(PROG2 <s1> <s2> ... <sN>) [FSUBR]

évalue séquentiellement les différentes expressions <s1> <s2> ... <sN>. PROG2 ramène en valeur de la deuxième évaluation (i.e. celle de <s2>).

```
ex : (PROG2 (PRIN1 1)(PRIN1 2)(PRIN1 3))  1 2 3  ☞  2
```

(PROGN <s1> ... <sN>) [FSUBR]

évalue les différentes expressions <s1> ... <sN>. PROGN ramène la valeur de la dernière évaluation (i.e. celle de <sN>).

```
ex : (PROGN (PRIN1 1)(PRIN1 2)(PRIN1 3))  1 2 3  ☞  3
```

(POUR <at> <s1> ... <sN>) [FSUBR]

si l'atome littéral <at> est l'atome EVAL, l'interprète va évaluer les différentes expressions <s1> ... <sN> à la manière d'un PROGN et POUR va donc ramener la valeur de <sN>. En revanche si l'atome <at> n'est pas EVAL, l'interprète va se désintéresser complètement du reste des expressions et ramener la valeur NIL.
Cette fonction permet donc de ne faire évaluer des expressions que par des programmes bien définis. Ces programmes doivent interpréter toutes les expressions lues en filtrant les POUR qui les intéressent.

Les POUR actuellement interceptés sont :
- EVAL par l'interprète standard
- PRETTY par le PRETTY-PRINT et le CROSS-REFERENCE
- INDEX par l'indexeur
- COMPIL par le compilateur

ex : l'occurence de
   (POUR PRETTY (STATUS 6 8))
 dans un fichier indique au PRETTY-PRINT et à lui-seul

d'imprimer tous les nombres en octal.

(QUOTE <s>) [FSUBR]

ramène la S-expression <s> non-évaluée. Cette fonction est utilisée comme argument des fonctions de type SUBR dont on ne désire pas évaluer les arguments. Il existe un macro-caractère standard qui facilite cette écriture, la caractère quote (l'apostrophe) ' .

```
ex : (QUOTE (ADD1 4))    ☞  (ADD1 4)
     '(A (B C))          ☞  (A (B C))
     'A                  ☞  A
     ''A                 ☞  'A
```

(FUNCTION <s>) [FSUBR]

est équivalent à QUOTE. FUNCTION a été ajouté par souci de compatibilité avec d'autres systèmes LISP qui utilisent cette fonction comme déclaration pour le compilateur, mais dont l'effet est identique à la fonction QUOTE.

```
ex : (FUNCTION (LAMBDA (X) X))  ☞  (LAMBDA (X) X)
```

(ID <s>) [SUBR à 1 argument]

c'est la fonction identité : ID ramène l'expression <s> qui est évaluée.

```
ex : (ID 'A)         ☞  A
     (ID (ADD1 3))   ☞  4
```

(COMMENT ...) [FSUBR]

est une fonction qui permet d'introduire des commentaires dans une définition de fonction. La valeur de cette fonction est toujours l'atome COMMENT. ATTENTION : cette fonction ne peut être placée qu'en MILIEU de PROGN pour ne pas perturber les évaluations.
Dans la pratique les commentaires seront introduits, en tous lieux, encadrés du délimiteur spécial point-virgule.

```
ex : (COMMENT)                        ☞  COMMENT
     (COMMENT et voila pourquoi)      ☞  COMMENT
```

(LAMBDA <s> <s1> ... <sN>) [FSUBR]

la valeur d'une forme LAMBDA est cette forme elle-même. Cette nouvelle fonction a été rajoutée pour éviter de "quoter" les lambda-expressions explicites des fonctionnelles.

```
ex : (LAMBDA (X) X)      ☞  (LAMBDA (X) X)
     (MAPC '(A B C) (LAMBDA (X) (PRIN1 X))) A B C  ☞  NIL
```

(LASTCALL <n>) [SUBR à 1 argument]

si <n> n'est pas fourni, ramène en valeur le corps de la dernière fonction appelée dynamiquement (i.e. la LAMBDA-expression qui contient l'appel de LASTCALL) sous la forme d'une LAMBDA-expression. Si <n> est

donné, LASTCALL ramène le corps du <n>ième dernier appel. S'il n'y a pas eu au moins <n> appels, LASTCALL ramène NIL. Cette fonction est utilisée pour réaliser des traces (voir le chapitre sur les traces).

```
ex : ((LAMBDA () (PRINT (LASTCALL))))
     (LAMBDA NIL (PRINT (LASTCALL)))
     ☞  (LAMBDA NIL (PRINT (LASTCALL)))

     (DE FOO (L) (FUU L) 'OK)    ☞  FOO

     (DE FUU (L)
       (PRINT '** 1 '= (LASTCALL 1))
       (PRINT '** 2 '= (LASTCALL 2))
      L)   ☞  FUU

     (FUU 6)
     ** 1 = (LAMBDA (L) (PRINT '** 1 (LASTCALL 1)) (PRINT
            '** 2 (LASTCALL 2))
     ** 2 = (LAMBDA (L) (FUU L) 'OK)
     ☞  6
```

(SELF <s1> ... <sN>) [SUBR à N arguments]

appelle la dernière lambda-expression utilisée dynamiquement avec <s1> ... <sN> pour arguments. Cette fonction permet de résoudre le problème des anciennes fonctions LABEL sans avoir à nommer obligatoirement les lambda-expressions explicites récursives. Si SELF est employée dans un mauvais contexte (i.e. à l'extérieur d'une LAMBDA-expression), une erreur apparait dont le libellé est :

** SELF error.

SELF pourrait être défini en VLISP de la manière suivante :

```
(DF SELF (L)
   (IF (LASTCALL 2)
      (APPLY (LASTCALL 2) (EVLIS L))
      (ERROR "SELF error.")))
```

```
ex : ((LAMBDA (L)
        (IF (NULL (CDR L))
            (CAR L)
            (SELF (CDR L))))
      '(A B C D))
     ☞  D
```

3.2 Les Prédicats De Base

3.2.1 Tests Sur Les Types -

(ATOM <s>) [SUBR à 1 argument]

teste si <s> est un atome, i.e. un atome littéral, un nombre ou une chaîne. ATOM ramène T si ce test est vérifié, NIL dans le cas contraire.

```
ex : (ATOM 'ARGH)      ☞  T
     (ATOM 42)         ☞  T
     (ATOM "OUPS")     ☞  T
     (ATOM '(A B))     ☞  NIL
```

(LITATOM <s>) [SUBR à 1 argument]

teste si <s> est un atome littéral. LITATOM ramène T si le test est vérifié et NIL dans le cas contraire.

```
ex : (LITATOM 'ARGH)  ☞ T
     (LITATOM 43)     ☞ NIL
     (LITATOM "OUPS") ☞ NIL
     (LITATOM '(A B)) ☞ NIL
```

(LISTP <s>) [SUBR à 1 argument]

teste si <s> est une liste. LISTP ramène T si le test est vérifié et NIL dans le cas contraire.

```
ex : (LISTP 'ARGH)    ☞ NIL
     (LISTP 44)       ☞ NIL
     (LISTP "OUPS")   ☞ NIL
     (LISTP '(A B))   ☞ T
```

(NULL <s>) [SUBR à 1 argument]

teste si <s> est égal à NIL. NULL ramène T si le test est vérifié et NIL dans le cas contraire.

```
ex : (NULL NIL)  ☞ T
     (NULL T)    ☞ NIL
```

(NOT <s>) [SUBR à 1 argument]

cette fonction est identique à NULL.

3.2.2 Les Comparaisons -

(BOUNDP <at>) [SUBR à 1 argument] {pour BOUNDed Predicate}

teste si l'atome littéral <at> donné en argument possède une valeur (plus précisement possède une C-valeur différente de UNDEF). Ce prédicat est utlisé pour tester des variables libres sans crainte de l'erreur ** UNDEFINED VARIABLE.

```
ex :  (SETQ XX 33)     ☞ 33
      (BOUNDP 'XX)     ☞ T
      (BOUNDP '?co?)   ☞ NIL
                (si ?co? apparait pour la premiere fois)
```

(EQP <s1> <s2>) [SUBR à 2 arguments] {pour EQ Pointer}

sert à tester 2 atomes litéraux. EQP ramène T si <s1> et <s2> sont les mêmes atomes litéraux, sinon EQP ramène NIL. Si <s1> et <s2> sont des nombres, des chaînes ou des listes, EQP teste si <s1> et <s2> ont la même adresse physique.

```
ex : (EQP 'A (CAR '(A)))    ☞ T
     (EQP '(A B) '(A B))    ☞ NIL
```

(NEQP <s1> <s2>) [SUBR à 2 arguments] {pour Not EQ Pointer}

est équivalent à (NOT (EQP <s1> <s2>)).

(EQ <s1> <s2>) [SUBR à 2 arguments]

sert à tester 2 atomes (de n'importe quel type : atome littéral, nombre ou chaîne). EQ ramène T si les 2 atomes <s1> et <s2> sont égaux et NIL s'ils ne le sont pas.
Rappelons que :
- 2 atomes litéraux sont égaux s'ils ont le même nom externe
- 2 nombres sont égaux s'ils ont la même valeur.
- 2 chaînes sont égales si elles ont la même longeur et contiennent les mêmes caractères.

Si <s1> et <s2> sont des listes, EQ teste si <s1> et <s2> ont la même adresse physique (de la même manière que la fonction EQP).

```
ex : (EQ 'A (CAR '(A)))      ☞ T
     (EQ (ADD1 119) 120)     ☞ T
     (EQ "STRR" "STRR")      ☞ T
     (EQ '(A B) '(A B))      ☞ NIL
```

(NEQ <s1> <s2>) [SUBR à 2 arguments] {pour Not EQ}

est equivalent à (NOT (EQ <s1> <s2>)).

(EQUAL <s1> <s2>) [SUBR à 2 arguments]

est la fonction de comparaison la plus générale. EQUAL teste si <s1> et <s2> ont la même structure. Si le test est verifié, EQUAL ramène T dans le cas contraire EQUAL ramène NIL.

EQUAL peut se définir en VLISP :

```
(DE EQUAL (l1 l2)
   (COND ((ATOM l1) (EQ l1 l2))
         ((ATOM l2) NIL)
         (T (AND (EQUAL (CAR l1) (CAR l2))
                 (EQUAL (CDR l1) (CDR l2))))))
```

```
ex : (EQUAL '(A (B . C) D) '(A (B . C) D))   ☞ T
```

(NEQUAL <s1> <s2>) [SUBR à 2 arguments]

est équivalent à (NOT (EQUAL <s1> <s2>))

3.3 Les Fonctions De Contrôle

3.3.1 Les Fonctions De Contrôle De Base -

(OR <s1> ... <sN>) [FSUBR]

évalue successivement les différentes expressions <s1> ... <sN> jusqu'à ce que l'une de ces évaluations ait une valeur différente de NIL. OR ramène cette valeur.

ex : (OR NIL NIL 2 3) ☞ 2

(AND <s1> ... <sN>) [FSUBR]

évalue successivement les différentes expressions <s1> ... <sN>. Si la valeur d'une de ces évaluations est égale à NIL, AND ramène NIL sinon AND ramène la valeur de la dernière évaluation <sN>. AND permet de construire une structure de contrôle de type :
    si <s1> alors si <s2> alors ... <sN>

```
ex : (AND 1 2 3 4)    ☞ 4
     (AND 1 2 () 4)   ☞ NIL
```

(IF <s1> <s2> <s3> ... <sN>) [FSUBR]

si la valeur de l'évaluation de <s1> est différente de NIL, IF ramène la valeur de l'évaluation de l'expression <s2>, sinon IF évalue en séquence les différentes expressions <s3> ... <sN> et ramène la valeur de la dernière évaluation <sN>. IF permet de construire une structure de contrôle de type :
    si <s1> alors <s2> sinon <s3> ... <sN>

```
ex : (IF T 1 2 3)     ☞ [illegible]
     (IF NIL 1 2 3)   ☞ [illegible]

     (DE ACK (x y)
        (IF (= x 0)
           (1+ y)
           (ACK (IF (= y 0)
                    1
                    (ACK x (1- y))))))
```

(IFN <s1> <s2> <s3> ... <sN>) [FSUBR] {pour IF Not}

si la valeur de l'évaluation de <s1> est égale à NIL, IFN ramène la valeur de l'évaluation de l'expression <s2>, sinon IFN évalue en séquence les différentes expressions <s3> ... <sN> et ramène la valeur de la dernière évaluation <sN>. IFN permet de construire une structure de contrôle de type :
    si non <s1> alors <s2> sinon <s3> ... <sN>

```
ex : (IFN T 1 2 3)     ☞ 3
     (IFN NIL 1 2 3)   ☞ 1
```

(COND <l1> ... <lN>) [FSUBR]

est l'instruction conditionnelle la plus générale. Les différents arguments <l1> ... <lN> sont des listes appelées clauses qui ont la structure suivante :

(<ss> <s1> ... <sN>)
COND va sélecter une seule de ces clauses : celle dont la valeur de l'évaluation de son premier élément <ss> est différente de NIL. COND évalue alors en séquence les différentes expressions <s1> ... <sN> et ramène la valeur de la dernière évaluation <sN>. Si la clause sélectée n'a qu'un élément <ss>, COND ramène la valeur de l'évaluation de <ss> (i.e. la valeur qui a déclenché la sélection de cette clause). COND permet de construire des structures de contrôle de type :
si ... alors ... sinon si ... alors ... ...
Si aucune clause n'est sélectée, COND ramène NIL.

donc :

```
(COND                 est equivalent à   (COND
  (p1 e11 e12 e13)                          (p1 (PROGN e11 e12 e13))
  (p2 e21 e22)                              (p2 (PROGN e21 e22))
  (p3)                                      ((SETQ aux p3) aux)
  (p4 e41))                                 (p4 e41))
```

ex : (COND (NIL 1 2) (T 3 4 5)) ☞ 5

```
(COND ((ZEROP X) 'ZERO)
      ((ODDP X)  'IMPAIR)
      ((EVENP X) 'PAIR)
      (T 'What?!?))
```

(SELECT <s> <l1> ... <lN> <lf>) [FSUBR]

comme pour la fonction COND, SELECT va sélecter une des clauses <l1> ... <lN>. Le sélecteur de ces clauses est la valeur de l'évaluation de <s>, la sélection s'effectue par comparaison (au moyen du prédicat EQUAL) du sélecteur avec la valeur de l'évaluation du premier élément de chacunes des clauses <l1> ... <lN>. Dès qu'une clause est sélectée, SELECT évalue séquentiellement le reste de la clause et ramène la valeur de la dernière évaluation. Si aucune des clauses <l1> ... <lN> n'est sélectée, la dernière clause <lf> est évaluée; SELECT ramène la valeur de (PROGN <lf>). SELECT permet donc de construire facilement des aiguillages.

ex : (SELECT (ADD1 9)
        (12 'BOF)
        ((SUB1 11) 'OUI)
        ('NAN))
     ☞ OUI

     (SELECT (PLUS 5 5)
        ('ZZ 'BOF)
        ('(A B) 'HUM)
        ('NAN))
     ☞ NAN

(SELECTQ <s> <l1> ... <lN> <lf>) [FSUBR]

comme pour la fonction COND, SELECTQ va sélecter une des clauses <l1> ... <lN>. Le sélecteur de ces clauses est la valeur de l'évaluation de <s>, la sélection s'effectue par comparaison du sélecteur avec :
- le CAR (non évalué) de la clause si ce CAR est atomique (en utilisant le prédicat EQ)
- les différents éléments du CAR (non évalué) de cette clause en utilisant la fonction MEMBER. L'utilisation de cette fonction permet d'avoir des sélecteurs de type liste.
Dès qu'une clause est sélectée, SELECTQ évalue en séquence le reste de la

clause et ramène la valeur de la dernière évaluation.
Si aucune des clauses <l1> ... <lN> n'est sélectée, SELECTQ évalue automatiquement la dernière clause <lf> et ramène la valeur de (PROGN <lf>). SELECTQ permet donc de construire des aiguillages sur valeurs constantes.

```
ex : (SELECTQ 'ROUGE
        (VERT 'ESPOIR)
        ((BLEU ROUGE JAUNE) 'OK)
        ('NON))
     ☞ OK

     (SELECTQ 'BLANC
        (VERT 'ESPOIR)
        ((BLEU ROUGE JAUNE) 'OK)
        ('NON))
     ☞ NON
```

(WHILE <s> <s1> ... <sN>) [FSUBR]

tant que la valeur de l'évaluation de <s> est différente de NIL, WHILE va évaluer en séquence les différentes expressions <s1> ... <sN>. WHILE ramène toujours NIL en valeur (qui est la dernière évaluation de <s> qui fait sortir de la boucle WHILE). Cette fonction permet de construire des boucles conditionnelles d'une manière fort commode, ainsi que des boucles infinies en utilisant la forme : (WHILE T ...).

WHILE est équivalent à la forme VLISP :

```
(LET () (IFN <s> NIL <s1> ... <sN> (SELF)))
```

```
ex : (SETQQ L (A B C D))                          ☞ (A B C D)
     (WHILE L (PRIN1 (NEXTL L))) A B C D          ☞ NIL
```

(UNTIL <s> <s1> ... <sN>) [FSUBR]

tant que la valeur de l'évaluation de <s> est égale à NIL, UNTIL va évaluer en séquence les différentes expressions <s1> ... <sN>. UNTIL ramène la valeur de l'évaluation de <s> qui a fait arrêter la boucle. A la valeur ramenée près, UNTIL est l'équivalent de (WHILE (NOT <s>) <s1> ... <sN>).

```
ex : (SETQQ L (A B 2 C D))  ☞ (A B 2 C D)
     (UNTIL (NUMBP (CAR L)) (PRIN1 (NEXTL L))) A B  ☞ 2
```

(REPEAT <s> <s1> ... <sN>) [FSUBR]

évalue <s>. Cette valeur doit être un nombre <n>. REPEAT évalue alors <n> fois les différentes expressions <s1> ... <sN>. Si l'évaluation de <n> n'est pas un nombre strictement positif, la boucle n'est pas exécutée et REPEAT ramène NIL, sinon REPEAT ramène la dernière évaluation de <sN>.

```
ex : (REPEAT 10 (PRIN1 '*)) * * * * * * * * * *   ☞ *
     (REPEAT 0 (PRIN1 '*))  ☞ NIL
```

(REPEATUNTIL <s1> <s2> ... <sN> <s>) [MACRO]

correspond à une boucle WHILE avec le test en fin de corps de boucle. REPEATUNTIL correspond donc à la forme :
(WHILE (PROGN <s1> <s2> ... <sN> <s>))

ATTENTION : cette MACRO se trouve dans le fichier SYS:VLISP.INI (cf: l'Appendice A) qui est normalement chargé à l'appel de VLISP (voir le chapitre Configuration).

3.3.2 Les Fonctions D'échappement -

(LESCAPE <s1> ... <sN>) [FSUBR] {pour Lambda ESCAPE}

évalue les différentes expressions <s1> ... <sN> en séquence et ramène la valeur de la dernière évaluation <sN>. De plus LESCAPE fait sortir de la dernière lambda-expression connue (fonction utilisateur ou lambda-expression explicite) avec pour valeur la valeur du LESCAPE.
Si cette fonction n'est pas utilisée à l'intérieur d'une lambda-expression (par exemple dans le corps d'un ESCAPE, d'un PROG, d'un DO ou même au top-level) une erreur apparait dont le libellé est :
** LESCAPE ERROR.

```
Ex : (LET ((n 5) (l NIL))
        (WHILE T
           (IF (ZEROP n)
              (LESCAPE l)
              (NEWL l (DECR n)))))
      ☞  (0 1 2 3 4)
```

(ESCAPE <at> <s1> ... <sN>) [FSUBR]

est la fonction de contrôle la plus puissante des systèmes VLISP. <at> est un atome littéral qui devient le nom d'une fonction d'échappement, puis les différentes expressions <s1> ... <sN> sont évaluées en séquence. Si au cours de ces évaluations une forme de type (<at> <ss1> ... <ssN>) est rencontrée, les différentes expressions <ss1> ... <ssN> sont évaluées et ESCAPE ramène la valeur de la dernière évaluation (i.e. celle de <ssN>). Si une telle forme n'est pas rencontrée, ESCAPE ramène la valeur de l'évaluation de <sN>.

```
ex : (ESCAPE EXIT
        (MAPC '(A B 2 C)
          (LAMBDA (X) (AND (NUMBP X) (EXIT 'OUI))))
        'NON)
       ☞  OUI
```

(ESCLOOP <at> <s1> ... <sN>) [MACRO]

est une macro qui permet de fabriquer des boucles contrôlées par une fonction d'échappement.

(ESCLOOP <at> <s1> ... <sN>) correspond à l'appel :

```
(ESCAPE <at>
   (WHILE T <s1> ... <sN>))
```

ESCLOOP est donc identique à une fonction d'échappement ESCAPE dont le

corps est évalué tant qu'il n'y a pas d'appel de la fonction d'échappement <at>.

ATTENTION : cette MACRO se trouve dans le fichier SYS:VLISP.INI (cf: l'Appendice A) qui est normalement chargé à l'appel de VLISP (voir le chapitre Configuration).

3.3.3 Les Fonctions De Contrôle De Type PROG -

Ce type de structure de contrôle permet à l'usager conservateur d'écrire des séquences VLISP sans structure, vertu nocturne partagée avec certains autres langages.
On peut se "brancher à des étiquettes" (fonctions GO et GOTO), sortir d'un corps de PROG comme en FORTRAN (avec la fonction éternelle : RETURN), et faire des "boucles" comme en ALGOL avec la fonction DO.
Ce type de structure de contrôle a été conservé dans un souci de compatibilité avec certains autres systèmes LISP qui ne possèdent que ce type de structure de contrôle, les fonctions d'échappement des systèmes VLISP étant à la fois plus puissantes et plus rapides à l'interprétation.

(PROG <l> <s1> ... <sN>) [FSUBR]

<l> est une liste d'atomes litéraux qui servent de variables locales à l'interieur du PROG. Elles sont liées à la valeur NIL à l'entrée du PROG et restaurées à leurs anciennes valeurs au retour du PROG. Cette protection des variables locales ne porte que sur leurs C-valeurs. Cette liste de variables locales peut être vide mais ne peut pas être omise. <s1> ... <sN> est une liste de formes qui sont évaluées en séquence. Si dans cette liste se trouvent des atomes, ils sont considérés comme des étiquettes et ne sont donc pas évalués. La valeur d'un PROG, s'il n'est pas interrompu par un RETURN, est la valeur de l'évaluation de <sN>.

(GO <a>) [FSUBR]

<a> est un atome de n'importe quel type (atome littéral, nombre ou chaîne) et doit être le nom d'une étiquette existante d'un PROG actif. L'évaluation de la forme (GO <a>) fait reprendre l'évaluation à la forme qui suit l'étiquette <a>. Si <a> n'est pas une étiquette connue ou si la forme (GO <a>) apparait à l'exterieur de tout PROG, une erreur apparait dont le libellé est :
** LABEL ERROR : <a>
dans lequel le nom de l'étiquette incriminée est imprimé.

(GOTO <s>) [SUBR à 1 argument]

est identique à la fonction GO mais l'argument <s> est évalué et doit ramener une etiquette en valeur sous peine de déclencher l'erreur LABEL ERROR.

(RETURN <s>) [SUBR à 1 argument]

sort du PROG courant. La valeur du PROG est celle de l'évaluation de <s>. Si une forme (RETURN <s>) apparait dynamiquement en dehors d'un PROG, une erreur apparait dont le libellé est :
** RETURN ERROR.

(DO (<at> <si> <sr>) (<p> <r1> ... <rN>) <s1> ... <sN>) [FSUBR]

permet de construire des boucles de manière aisée, avec une (ou plusieurs) variable de contrôle.
Cette variable de contrôle <at> est équivalente à une variable locale de PROG mais est initialisée à l'entrée du DO avec la valeur de l'évaluation de <si> , et après chaque itération (i.e. chaque évaluation de (PROGN <s1> ... <sN>)) elle prend la nouvelle valeur <sr>. La boucle s'arrête quand la condition d'arrêt <p> est vérifiée et le DO ramène la valeur de l'évaluation de (PROGN <r1> ... <rN>) i.e. la valeur de <rN>.

l'appel :

```
(DO (I 0 (ADD1 I))
      ((EQ I 10) (TERPRI) 'OK)
      (PRIN1 I))
0 1 2 3 4 5 6 7 8 9 10
☞  OK
```

est équivalent à :

```
(PROG (I)
   (SETQ I 0)
   (UNTIL
      (PROGN (PRIN1 I) (EQ I 10))
      (SETQ I (ADD1 I)))
   (TERPRI)
   'OK)
```

Il existe une deuxième forme de DO comportant plusieurs variables de contrôle, la syntaxe du 1er argument devient :
((at1 si1 sr1) ... (atN siN srN))
la forme du DO devient donc :

```
(DO  ((<at1> <si1> <sr1>) ... (<atN> <siN> <srN>))
     (<p> <r1> ... <rN>)
     <s1> ... <sN>)
```

Dans ce cas les différentes valeurs d'initialisation <si1> ... <siN> sont évaluées dans le contexte de l'appelant (i.e. avant toute liaison de variables de contrôle), et les variables sont initialisées en PARALLELE . De même les différentes valeurs d'itération sont évaluées avant toute nouvelle affectation, et les variables sont re-affectées en PARALLELE.

Tout comme dans un PROG, il peut y avoir des étiquettes dans le corps du DO et il est possible de retourner d'un DO en utilisant l'instruction RETURN.

```
ex :  (DE MAPL (fn x y)
        (DO  ((x x (CDR x))
              (y y (CDR y))
              (z () (cons (fn (CAR x) (CAR y)) z)))
             ((OR (NULL x) (NULL y))
              (FREVERSE z))))

      (MAPL 'CONS '(A B C) '(D E F G))  ☞  ((A . D) (B . E) (C . F))
```

Formes dégénérées de DO :

(DO () () <s1> ... <sN>) est équivalent à
(PROGN <s1> ... <sN>) s'il n'y a ni étiquettes ni GO ni GOTO

```
   (DO () (NIL) <s1> ... <sN>)        est équivalent à
   (UNTIL () <s1> ... <sN>)
ou (WHILE T  <s1> ... <sN>)   même remarque que précédement

   (DO (x) () <s1> ... <sN>)          est équivalent à
   (PROG (x)  <s1> ... <sN>)
```

(CYCLE) [FSUBR]

permet de sortir d'un corps de DO et de revenir au niveau du test d'itération. ATTENTION : CYCLE est différent du RETURN qui fait sortir de la fonction DO entiérement.

Une fonction DO possédant des appels de CYCLE peut être décrite au moyen d'un PROG de la manière suivante :

```
(PROG (at1 ... atN)
   (SETQ at1  si1
         ...  ...
         atN  siN)
   (UNTIL (PROGN (ESCAPE CYCLE s1 ... sN) p)
      (SETQ at1  sr1
            ...  ...
            atN  srN))
   s1
   ...
   sN)
```

Si une forme (CYCLE) apparait dynamiquement en dehors d'un DO, une erreur apparait dont le libellé est :
** CYCLE ERROR.

3.3.4 Les Fonctionnelles -

Toutes ces fonctions sont de type SUBR et utilisent des fonctions <fn> en argument. Ces fonctions peuvent être de n'importe quel type (SUBR , EXPR ...) mais seuls leurs premiers arguments seront liés à des objets variables, tous les autres seront liés a NIL.

(SOME <l> <fn>) [SUBR à 2 arguments]

applique successivement la fonction <fn> à tous les éléments de la liste <l> jusqu'à ce qu'une de ces applications livre une valeur différente de NIL qui est ramenée en valeur.

```
ex : (SOME '(A B 2 C) 'NUMBP)  ☞  2
     (SOME '(A B 2 C) 'LISTP)  ☞  NIL
```

(EVERY <l> <fn>) [SUBR à 2 arguments]

applique successivement la fonction <fn> à tous les éléments de la liste <l>. Si l'une de ces applications livre une valeur égale à NIL, EVERY ramène NIL, sinon EVERY ramène la valeur de la dernière application.

```
ex : (EVERY '(A B 3 C) 'ATOM)     ☞  T
     (EVERY '(A B 3 C) 'LITATOM)  ☞  NIL
```

(ORF <s> <fn1> ... <fnN>) [SUBR à N arguments]

applique les différentes fonctions <fn1> ... <fnN> à l'argument <s> jusqu'à ce que l'une de ces applications livre une valeur différente de NIL qui est ramenée en valeur, sinon ORF ramène NIL.

```
ex : (ORF  3 'LITATOM 'NUMBP 'LISTP)  ☞  3
     (ORF  3 'LITATOM 'LISTP)         ☞  NIL
```

(ANDF <s> <fn1> ... <fnN>) [SUBR à N arguments]

applique les différentes fonctions <fn1> ... <fnN> à l'argument <s>. Si l'une de ces applications livre une valeur égale à NIL, ANDF ramène NIL sinon ANDF ramène la valeur de la dernière application (i.e. celle de (<fnN> <s>)).

```
ex : (ANDF 3 'GZP 'ODDP)   ☞  3
     (ANDF 4 'GZP 'ODDP)   ☞  NIL
```

(MAPxxx <l> <fn>) [SUBR à 2 arguments]

Elles permettent d'appliquer la fonction <fn> sur certains sous-ensembles de la liste <l>.

applique la fonction <fn> sur :			ramène en valeur
<l> puis sur ses CDR successifs	les CAR successifs de <l>	<l> puis sur toutes ses sous-structures	
MAP	MAPC	MAPS	NIL
MAPLIST	MAPCAR	MAPSUB	la liste des valeurs de toutes les applications
MAPT	MAPCT	MAPST	la liste des valeurs différentes de NIL de toutes les applications

Les fonctions de type MAPS (elles n'existent qu'en VLISP) permettent d'explorer des structures arborescentes. Elles opèrent en ordre préfixe sur les CAR.

```
ex : (MAP  '(A (B C) D) 'PRINT)
     (A (B C) D)
     ((B C) D)
     (D)
     ☞  NIL

     (MAPC '(A (B C) D) 'PRINT)
     A
     (B C)
     D
     ☞  NIL

     (MAPS '(A (B C) D) 'PRINT)
```

```
(A (B C) D)
A
((B C) D)
(B C)
B
(C)
C
(D)
D
☞ NIL
```

(LIT <l> <s> <fn>) [SUBR à 3 arguments]

applique la fonction <fn>, à 2 arguments, en utilisant d'une part le CAR de la liste <l> et d'autre part le résultat de l'application de cette même fonction avec le CDR de la liste si la liste n'est pas vide ou bien l'expression <s> si la liste est vide. Le francais n'étant vraiment pas récursif, voici la description de cette fonction en VLISP :

```
(DE LIT (l s fn)
   (IF (NULL l)
       s
       (fn (CAR l) (LIT (CDR l) s fn))))
```

LIT permet d'écrire des définitions très élégantes et concises :

```
(APPEND x y)     =   (LIT x y 'CONS)
(MAPCAR l fn)    =   (LIT l () (LAMBDA (x y) (CONS (fn x) y)))
```

3.4 Les Fonctions De Recherche

3.4.1 Les Recherches Sur Des Objets VLISP -

(CAR <s>) [SUBR à 1 argument]

si <s> est un atome littéral, ramène sa C-valeur (i.e. la valeur associée à cet atome considéré comme une variable).
si <s> est une liste, ramène son premier élément.
Le CAR d'un nombre ou d'une chaîne est indéterminé.

```
ex : (CAR '(A B C))       ☞  A
     (SETQ X '(U P))      ☞  (U P)
     (CAR 'X)             ☞  (U P)
```

(CDR <s>) [SUBR à 1 argument]

si <s> est un atome littéral, ramène sa P-liste.
si <s> est une liste, ramène cette liste sans son premier élément.
Le CDR d'un nombre ou d'une chaîne est indéterminé.

```
ex : (CDR '(A B C))         ☞  (B C)
     (PUT 'X '(U P) 'I)     ☞  X
     (CDR 'X)               ☞  (I (U P))
```

(C....R <s>) [SUBR à 1 argument]

les 28 combinaisons de CAR et de CDR imbriqués sont disponibles.
(CADR <s>) est équivalent à (CAR (CDR <s>))
(CDAAR <s>) est équivalent à (CDR (CAR (CAR <s>)))
(CADDDR <s>) est équivalent à (CAR (CDR (CDR (CDR <s>))))

(MEMQ <a> <l>) [SUBR à 2 arguments]

si l'atome <a> est un élément de la liste <l>, ramène la partie de la liste <l> commençant à l'atome <a>, sinon ramène NIL. Cette fonction utilise le prédicat EQ pour tester la présence de l'atome <a> dans la liste.

```
ex : (MEMQ 'C '(A B C D E))    ☞  (C D E)
     (MEMQ 'Z '(A B C D E))    ☞  NIL
```

(MEMBER <s> <l>) [SUBR à 2 arguments]

si l'expression quelconque <s> est un élément de la liste <l> ramène la partie de <l> commençant à l'expression <s>, sinon ramène NIL. Cette fonction est identique à la fonction MEMQ mais utilise le prédicat EQUAL.

```
ex : (MEMBER '(B C) '((A (B C)) (B C) D))   ☞  ((BC) D)
```

(NTH <n> <l>) [SUBR à 2 arguments]

ramène la partie de la liste <l> commençant à son <n>ième élément. Si <l> n'est pas une liste ou si (LENGTH <l>) est plus petit que <n>, NTH ramène NIL. Si <n> <= 1, NTH ramène la liste <l> en entier.

NTH peut être défini en VLISP de la manière suivante :

```
(DE NTH (n l)
   (IF (LE n 1)
       l
       (NTH (SUB1 n) (CDR l))))
```

```
ex : (NTH 3 '(A B C D E))   ☞  (C D E)
```

(CNTH <n> <l>) [SUBR à 2 arguments] {pour Car NTH}

ramène le <n>ième élément de la liste <l>. CNTH est équivalent à (CAR (NTH <n> <l>)).
L'appel de CNTH est implicite en cas de forme dont la fonction est un nombre (cf: l'interprète).

CNTH peut être défini en VLISP de la manière suivante :

```
(DE CNTH (n l)
   (IF (LE n 1)
       (CAR l)
       (CNTH (SUB1 n) (CDR l))))
```

```
ex : (CNTH 3 '(A B C D E F))    ☞  C
     ((ADD1 4) '(A B C D E F))  ☞  E
```

(LAST <l> <n>) [SUBR à 2 arguments]

ramène la liste ne contenant que le dernier élément de la liste <l>. Si un deuxième argument numérique lui est ajouté LAST livre la liste des <n> derniers éléments de la liste <l>. Cet ajout est très utile, et permet notammnent de détruire physiquement le ou les derniers éléments d'une liste.

LAST à 1 argument peut être défini en VLISP :

```
(DE LAST (l)
   (IF (ATOM (CDR l))
       l
       (LAST (CDR l))))
```

```
ex : (LAST '(A B C))             ☞  (C)
     (LAST '(A B C . D))         ☞  (C . D)
     (SETQ L '(A B C D))         ☞  (A B C D)
     (LAST L 2)                  ☞  (C D)
     (RPLACD (LAST L 2))         ☞  (C)
     L                           ☞  (A B C)
```

(LENGTH <s>) [SUBR à 1 argument]

si <s> est une liste, ramène le nombre d'éléments de cette liste. Si <s> n'est pas une liste, LENGTH ramène 0.

LENGTH peut être défini en VLISP :

```
(DE LENGTH (l)
   (IF (ATOM l)
       0
       (ADD1 (LENGTH (CDR l)))))
```

```
ex : (LENGTH '(A (B C) D E )  ☞  4
     (LENGTH "Esar")          ☞  0
```

3.4.2 Recherche De Type -

(TYPEP <s>) [SUBR à 1 argument] {pour TYPE Predicate}

ramène différents atomes en fonction du type de l'argument <s>. Cette fonction peut être utilisée pour construire des fonctions dites génériques. TYPEP ramène l'atome :
- LITATOM si <s> est un atome littéral,
- NUMBP si <s> est un nombre (pour connaitre le type de ce nombre, il faut utiliser la fonction TYPNUMB),
- STRINGP si <s> est une chaîne,
- LISTP si <s> est une liste,
- NIL si <s> n'est pas un objet VLISP (par exemple une adresse de lancement de SUBR, une adresse de tableau etc ...).

```
ex : (TYPEP 'A)      ☞  LITATOM
     (TYPEP 45.2)    ☞  NUMBP
     (TYPEP '(X Y)   ☞  LISTP
```

(TYPEFN <at>) [SUBR à 1 argument] {pour TYPE FuNction}

Cette fonction teste si l'atome littéral <at> donné en argument possède une définition de fonction. TYPEFN ramène l'atome :
- EXPR si <at> possède une définition de type EXPR,
- FEXPR si <at> possède une définition de type FEXPR,
- SUBR si <at> possède une définition de type SUBR,
- FSUBR si <at> possède une définition de type FSUBR,
- MACIN si <at> possède une définition de type MACIN,
- MACOUT si <at> possède une définition de type MACOUT,
- MACRO si <at> possède une définition de type MACRO,
- ARRAY si <at> possède une définition de type ARRAY,
- AUTOLOAD si <at> est une fonction de type AUTOLOAD,
- NIL si <at> ne possède pas de définition de fonction.

Si l'atome possède plusieurs définitions, TYPEFN ramène le type de celle qu'utiliserait EVAL si on évaluait une forme dont la fonction était <at>, c'est-à-dire la première définition trouvée sur la P-liste de <at>.

```
ex : (TYPEFN 'TYPEFN)    ☞  SUBR
     (TYPEFN 'COND)      ☞  FSUBR
     (TYPEFN 'QUOI)      ☞  NIL
```

(TYPNUMB <n>) [SUBR à 1 argument] {pour TYPe NUMBer}

cette fonction teste le type du nombre donné en argument et ramène l'atome :
- FIX si le nombre <n> est de type entier,
- FLOAT si le nombre <n> est de type flottant,
- NIL si l'argument <n> n'est pas un nombre.

```
ex : (TYPNUMB 45)        ☞  FIX
     (TYPNUMB 120.12)    ☞  FLOAT
     (TYPNUMB 'NUMBER)   ☞  NIL
```

3.5 Fonctions De Création De Listes

(CONS <s1> <s2>) [SUBR à 2 arguments]

construit une liste dont le premier élément est <s1> et le reste la liste <s2>. Si <s2> est un atome, CONS produit la paire pointée
(<s1> . <s2>).
Il existe une écriture abrégée de la fonction CONS qui utilise les caractères spéciaux crochets carrés :
[<s1> . <s2>]

```
ex : (CONS 'A '(B C))        ☞  (A B C)
     (CONS 1 2)              ☞  (1 . 2)
     '[A . B]                ☞  (CONS A B)
     [ '(X Y) . '(U P) ]     ☞  ((X Y) U P)
```

(XCONS <s1> <s2>) [SUBR à 2 arguments] {pour eXchange CONS}

est équivalent à (CONS <s2> <s1>). XCONS est donc un CONS qui possède ses deux arguments inversés. Cette fonction n'est pas très utile pour vous mais permet dans certains cas d'améliorer le code produit par le compilateur.

```
ex : (XCONS 1 2)         ☞  (2 . 1)
     (XCONS 'A '(B C))   ☞  ((B C) . A)
```

(NCONS <s>) [SUBR à 1 argument] {pour Nil CONS}

est équivalent à la foncion (CONS <s> NIL). Cette fonction est utilisée par le compilateur pour améliorer le code produit ainsi que pour la macro-génération des expressions entre crochets carrés; en effet une entrée de type [<s>] sera macrogénérée en (NCONS <s>) ce qui est plus performant que (CONS <s> NIL) ou que (LIST <s>).

```
ex : (NCONS 'A)       ☞  (A)
     (NCONS '(U P))   ☞  ((U P))
     '[L]             ☞  (NCONS L)
```

(MCONS <s1> <s2> ... <sN>) [SUBR à N arguments] {pour Multiple CONS}

permet le CONS multiple. L'appel (MCONS <s1> <s2> ... <sN-1> <sN>) correspond à (CONS <s1> (CONS <s2> ... (CONS <sN-1> <sN>) ...)). Cette écriture peut s'abréger en utilisant les caractères spéciaux crochets carrés :

```
    [ <s1> <s2> ... <sN-1> . <sN>]
```

```
ex : (MCONS 'A 'B 'C)            ☞  (A B . C)
     '[ L (ADD1 X) . (FOO L)]    ☞  (MCONS L (ADD1 X) (FOO L)]
```

(DCONS <s> <l>) [SUBR à 2 arguments] {pour Distributive CONS}

permet de CONSer l'expression <s> à tous les éléments de la liste <l>.

DCONS peut être défini en VLISP de la manière suivante :

```
(DE DCONS (S L)
    (MAPCAR L (LAMBDA (L) (CONS S L))))
```

```
ex : (DCONS 'A '(X (Y) ((Z)))   ☞  ((A . X) (A Y) (A (Z)))
```

(LIST <s1> ... <sN>) [SUBR à N arguments]

ramène la liste des valeurs des différentes expressions <s1> ... <sN>. En terme de CONS l'appel (LIST <s1> <s2> ... <sN-1> <sN>) est équivalent à (CONS <s1> (CONS <s2> ... (CONS <sN-1> (CONS <sN> NIL)) ...)).
Il existe une écriture abrégée de la fonction LIST qui utilise les caractères spéciaux crochets carrés :

```
    [ <s1> ... <sN> ]
```

```
ex : (LIST 'A 'B 'C)      ☞  (A B C)
     (LIST)               ☞  NIL
     '[45 (ADD1 X) L]     ☞  (LIST 45 (ADD1 X) L)
     [[]]                 ☞  (NIL)
```

(LINEAR <s1> ... <sN>) [SUBR à N arguments]

ramène la liste de tous les atomes des différentes expressions <s1> ... <sN>. LINEAR "aplatit" donc tous ses arguments.

```
ex : (LINEAR 'A '(B (C . D)) 5)  ☞  (A B C D 5)
```

(COPY <s>) [SUBR à 1 argument]

fabrique une nouvelle copie de toute la liste <l>. Pour cette fonction la copie s'effectue à tous les niveaux de l'arborescence.

COPY peut être défini en VLISP de la manière suivante :

```
(DE COPY (s)
   (IF (ATOM s)
       s
       (CONS (COPY (CAR s)) (COPY (CDR s)))))
```

```
ex : (COPY 'A)                ☞  A
     (COPY '(A (B (C (D))))  ☞  (A (B (C (D))))
```

Cette fonction ne traite pas les listes circulaires ou partagées. Pour cela il faut utiliser la fonction :

```
(DE CIRCOPY (l ;; d)
   (LET ((l l) (new))
      (COND
         ((ATOM l) l)
         ((CASSQ l d))
         (T (SETQ new [NIL] d [[l . new] . d])
            (RPLACA new (SELF (CAR l)))
            (RPLACD new (SELF (CDR l)))))))
```

(SUBST <s1> <s2> <l>) [SUBR à 3 arguments]

fabrique une nouvelle copie de toute la liste <l> en substituant à l'expression <s2> l'expression <s1> à chacune de ses occurences. Si les deux expressions sont les mêmes pointeurs physiques (i.e. si (EQP <s1> <s2>) = T), l'interprète se permet d'appeler la fonction COPY à la place de SUBST, l'exécution de la fonction COPY étant beaucoup plus rapide.

SUBST peut être défini en VLISP de la manière suivante :

```
(DE SUBST (s1 s2 l)
   (COND ((EQUAL l s2) s1)
         ((ATOM l) l)
         (T (CONS (SUBST s1 s2 (CAR l))
                  (SUBST s1 s2 (CDR l))))))
```

```
ex : (SUBST '(X Y Z) '(A B) '((A B) C (D (A B))))
     ☞  ((X Y Z) C (D (X Y Z)))
```

(OBLIST) [SUBR à 0 argument]

ramène la (très longue) liste de tous les atomes litéraux présents dans le système. Cette liste ne contient pas l'atome UNDEF (ce qui évite bien des erreurs ** UNDEFINED VARIABLE).

(REVERSE <s1> <s2>) [SUBR à 2 arguments]

ramène une copie inversée du premier niveau de la liste <s1>. Si le deuxième argument est fourni, il est ajouté au moyen d'un NCONC physique à la première liste inversée : l'appel correspond donc à (NCONC (REVERSE <s1>) <s2>).

REVERSE peut être défini en VLISP de la manière suivante :

```
(DE REVERSE (s1 s2)
   (IF (NULL s1)
       s2
       (REVERSE (CDR s1) (CONS (CAR s1) s2))))
```

ex : (REVERSE '(A (B C) D)) ☞ (D (B C) A)
(REVERSE '((X Y) (U P)) '(G O)) ☞ ((U P) (X Y) G O)

On prétend parfois que cette fonction fait le même travail :

```
(DE REV (l)
   (IF (NULL (CDR l))
       l
       (CONS (CAR (REV (CDR l)))
             (REV (CONS (CAR l)
                  (REV (CDR (REV (CDR l)))))))))
```

(APPEND <l> <s>) [SUBR à 2 arguments]

ramène la concaténation d'une copie du premier niveau de la liste <l> à l'expression <s>. Si <s> n'est pas fourni, (APPEND <l>) ramène simplement une copie du premier niveau de <l>.

APPEND peut être défini en VLISP de la manière suivante :

```
(DE APPEND (l s)
   (IF (NULL l)
       s
       (CONS (CAR l) (APPEND (CDR l) s))))
```

ex : (APPEND '(A B C)) ☞ (A B C)
(APPEND '(A B C) '(X Y)) ☞ (A B C X Y)

(APPEND1 <l> <s1> ... <sN>) [SUBR à N arguments]

ramène la concaténation d'une copie du premier niveau de la liste <l> à la liste [<s1> ... <sN>]. APPEND1 est équivalent à
(APPEND <l> [<s1> ... <sN>]).

ex : (APPEND1 '(A B) 'C 'D) ☞ (A B C D)

(DELQ <a> <l>) [SUBR à 2 arguments]

ramène une copie du premier niveau de la liste <l> dans laquelle toutes les occurences de l'atome <a> ont été enlevées. Cette fonction utilise le prédicat EQ.

DELQ peut être défini en VLISP de la manière suivante :

```
(DE DELQ (a l)
   (COND ((NULL l) NIL)
         ((EQ a (CAR l)) (DELQ a (CDR l)))
         (T (CONS (CAR l) (DELQ a (CDR l))))))
```

ex : (DELQ 'A '(Z A Y A (B A C))) ☞ (Z Y (B A C))

(DELETE <s> <l>) [SUBR à 2 arguments]

ramène une copie du premier niveau de la liste <l> dans laquelle toutes les occurences de l'expression <s> ont été enlevées. Cette fonction utilise le prédicat EQUAL.

```
ex : (DELETE '(X Y) '(Z (X Y) U (X Y) P))  ☞  (Z U P)
```

3.6 Les Fonctions De Modification

Toutes les fonctions qui vont être décrites doivent être utilisées conformément au mode d'emploi, pour éviter de placer le système dans un état de confusion dramatique car elles modifient physiquement les structures VLISP.

D'une manière générale il est très vivement souhaité de ne pas modifier :
- les constantes symboliques de l'interprète (NIL T LAMBDA ...) et d'une manière générale toutes les atomes de l'interprète.
- les nombres
- les chaînes de caractères

Pour ces fonctions l'argument <obj> représente soit un atome littéral soit une liste. Modifier le CAR d'un atome revient à changer sa C-valeur, modifier son CDR revient à modifier sa P-liste.

(RPLACA <obj> <s>) [SUBR à 2 arguments] {pour RePLACe cAr}

remplace le CAR de <obj> par <s>. Ramène le nouvel <obj> en valeur.

```
ex : (SETQ X '(A B))              ☞  (A B)
     (RPLACA 'X '(C))             ☞  X
     X                            ☞  (C)
     (RPLACA X 'D)                ☞  (D)
     X                            ☞  (D)
     (RPLACA '(A B C) '(X Y))     ☞  ((X Y) B C)
```

(RPLACD <obj> <s>) [SUBR à 2 arguments] {pour RePLACe cDr}

remplace le CDR de <obj> par <si>. Ramène le nouvel <obj> en valeur.

```
ex : (RPLACD '(A B C) '(X Y Z))  ☞  (A X Y Z)
```

(RPLACB <obj> <l>) [SUBR à 2 arguments] {pour RePLACe Both}

remplace le CAR de <obj> par le CAR de <l> et le CDR de <obj> par le CDR de <l>. Cette fonction correspond à :
   (PROGN (RPLACA <obj> (CAR <l>)) (RPLACD <obj> (CDR <l>)))
Cette fonction est souvent utilisée dans des MACRO pour modifier physiquement l'appel de MACRO lui-même.

```
ex : (SETQ L1 '(A B C))      ☞  (A B C)
     (SETQ L2 L1)            ☞  (A B C)
     (RPLACB L1 '(X Y))      ☞  (X Y)
     L2                      ☞  (X Y)
```

(SET <obj1> <s1> ... <objN> <sN>) [SUBR à N arguments]

remplace tous les CAR des <obj> par leur <s> correspondant. SET ramène en valeur <sN>. A la valeur ramenée près, (SET <obj> <s>) est équivalent à (RPLACA <obj> <s>).

```
ex : (SET '(A B C) '(X Y))    ☞  (X Y)
```

(SETQ <at1> <s1> ... <atN> <sN>) [FSUBR]

<at1> ... <atN> sont des atomes littéraux qui ne sont pas évalués ; <s1> ... <sN> sont des expressions quelconques qui seront évaluées. SETQ est la fonction d'affectation la plus utilisée : chaque atome <at> est initialisé avec l'expression correspondante <si>. SETQ ramène <sN> en valeur.

```
ex : (SETQ L1 '(A B C))      ☞  (A B C)
     (SETQ L2 L1)            ☞  (A B C)
     (SETQ L3 L2 L4 'FOO)    ☞  FOO
```

(SETQQ <at1> <s1> ... <atN> <sN>) [FSUBR]

est identique à la fonction SETQ toutefois les différentes valeurs <s1> ... <sN> ne sont pas évaluées. SETQQ ramène <sN> en valeur.

```
ex : (SETQQ L1 (A B C))      ☞ (A B  C)
     (SETQQ L1 L2)           ☞ L2
```

(SYNONYM <at1> <at2>) [SUBR à 2 arguments]

donne à l'atome littéral <at1> les mêmes indicateurs spéciaux (SUBR FSUBR ...) que l'atome littéral <at2>. Cette fonction permet de changer le nom d'une fonction standard. SYNONYM ramène <at1> en valeur.

```
ex : (SYNONYM 'KONSS 'CONS)   ☞  KONSS
     (KONSS 1 2)              ☞  (1 . 2)
     (CONS 1 2)               ☞  (1 . 2)
```

(NEXTL <at>) [FSUBR] {pour NEXT List}

<at> (qui n'est pas évalué) doit être un atome littéral dont la valeur doit être une liste. NEXTL ramène le CAR de cette liste en valeur et donne comme nouvelle valeur de <at> le CDR de son ancienne valeur. Cette fonction est trés utile pour "avancer dans une liste" qui est la valeur d'un atome.
Cette fonction correspond en VLISP à :

```
   (PROG1 (CAR <at>) (SETQ <at> (CDR <at>)))
```

```
ex : (SETQ A '(X Y Z))    ☞  (X Y Z)
     (NEXTL A)            ☞  X
     A                    ☞  (Y Z)
```

(NEWL <at> <s>) [FSUBR] {pour NEW List}

<at> (qui n'est pas évalué) doit être un atome littéral dont la valeur est une liste. NEWL place en tête de cette liste la valeur de <s> et ramène en valeur la nouvelle liste formée.

Cette fonction correspond en VLISP à :
(SETQ <at> (CONS <s> <at>)).
L'utilisation conjuguée des fonctions NEXTL et NEWL permet de construire des pile-listes.

```
ex : (SETQ A '(X Y Z))      ☞  (X Y Z)
     (NEWL A 'W)            ☞  (W X Y Z)
     A                      ☞  (W X Y Z)
```

(EXCH <at1> <at2>) [FSUBR]

<at1> et <at2> (qui ne sont pas évalués) doivent être des atomes litéraux. EXCH va échanger leurs deux valeurs et ramène la nouvelle valeur de <at2> (i.e. l'ancienne valeur de <at1>).
Cette fonction correspond en VLISP à :
(SETQ <at2> (PROG1 <at1> (SETQ <at1> <at2>)))

```
ex : (SETQ A 5 B 10)   ☞  10
     (EXCH A B)        ☞  5
     A                 ☞  10
```

(INCR <at>) [FSUBR]

<at> (qui n'est pas évalué) doit être un atome littéral dont la valeur est un nombre entier. INCR ajoute 1 à la valeur de <at> et ramène cette nouvelle valeur.
Cette fonction correspond en VLISP à :
(SETQ <at> (ADD1 <at>))

```
ex : (SETQ X 2)   ☞  2
     (INCR X)     ☞  3
     X            ☞  3
```

(DECR <at>) [FSUBR]

<at> (qui n'est pas évalué) doit être un atome littéral dont la valeur est un nombre entier. INCR retranche 1 à la valeur de <at> et ramène cette nouvelle valeur.
Cette fonction correspond en VLISP à :
(SETQ <at> (SUB1 <at>))

```
ex : (SETQ X 3)   ☞  3
     (DECR X)     ☞  2
     X            ☞  2
```

(NCONC <l1> <l2>) [SUBR à 2 arguments]

concatène physiquement les deux listes <l1> et <l2> (i.e. place dans le CDR du dernier élément de <l1> un pointeur sur la liste <l2>). NCONC ramène la nouvelle liste <l1> en valeur. Si <l1> et <l2> sont les mêmes pointeurs physiques, NCONC permet de construire des listes circulaires.

```
ex : (SETQ L1 '(A B C) L2 L1)   ☞  (A B C)
     (NCONC L1 '(D E F))        ☞  (A B C D E F)
     L2                         ☞  (A B C D E F)
     (NCONC L1 L1)              ☞  (A B C D E F A B C D E F A B ...
```

(NCONC1 <l> <s1> ... <sN>) [SUBR à N arguments]

est équivalent à (NCONC <l> [<s1> ... <sN>]). Cette fonction est très utile pour rajouter des éléments à une liste un à un.

ex : (NCONC1 '(A B C) 'D 'E 'F) ☞ (A B C D E F)

(FREVERSE <l>) [SUBR à 1 argument] {pour Fast REVERSE}

renverse physiquement (et rapidement) la liste <l>. Cette fonction doit être manipulée avec précaution car elle modifie physiquement des structures VLISP, mais elle est évidement beaucoup plus rapide que la fonction traditionnelle REVERSE.

FREVERSE peu être défini en VLISP de la manière suivante :

```
(DE FREVERSE (l ;; r)
   (IF l
       (FREVERSE (CDR l) (RPLACD r l))
       r))
```

```
ex :  (SETQ L1 '(A B C D E))  ☞ (A B C D E)
      (SETQ L2 (CDR L1))      ☞ (B C D E)
      (SETQ L3 (LAST L1))     ☞ (E)
      (FREVERSE L1)           ☞ (E D C B A)
      L2                      ☞ (B A)
      L3                      ☞ (E D C B A)
```

(SMASH <l>) [SUBR à 1 argument]

enlève physiquement le premier élément de la liste <l> en conservant la même adresse physique pour le premier élément de la liste <l>. Il y a donc recopie du deuxième élément de <l> dans le premier puis destruction du deuxième élément. SMASH ramène la nouvelle liste <l> en valeur. (SMASH <l>) est équivalent à (RPLACB <l> (CDR <l>)).

```
ex : (SETQ L1 '(A B C) L2 L1)   ☞  (A B C)
     (SMASH L2)                 ☞  (B C)
     L1                         ☞  (B C)
```

(ATTACH <l> <s>) [SUBR à 2 arguments]

ajoute physiquement en tête de la liste <l> l'élément <s> en conservant la même adresse physique pour le premier élément de la liste <l>. Il y a donc recopie du premier élément de <l> dans un doublet libre puis modification du doublet libéré par <s>. ATTACH ramène la nouvelle liste <l> en valeur.
(ATTACH <l> <s>) est donc equivalent à (RPLACB <l> (CONS <s> <l>)).

```
ex : (SETQ L1 '(Y Z) L2 L1)     ☞  (Y Z)
     (ATTACH 'X L2)             ☞  (X Y Z)
     L1                         ☞  (X Y Z)
```

3.7 Fonctions Sur Les A-listes

En VLISP 10 comme dans tous les VLISP, les A-listes (les listes d'association) sont des listes de listes qui possèdent la structure suivante :
((var1 . val1) (var2 . val2) ... (varN . valN))
Chaque élément est une liste dont le CAR est considéré comme une variable et le CDR comme sa valeur associée.
Pour toutes les fonctions qui vont être décrites, l'argument <al> doit être une A-liste.

3.7.1 Recherche Sur Les A-listes -

(ASSQ <a> <al>) [SUBR à 2 arguments]

ramène l'élément de la A-liste <al> dont le CAR (la variable) est égal à l'atome <a>, sinon ASSQ ramène NIL. Cette fonction utilise le prédicat EQ.

ASSQ peut être défini en VLISP de la manière suivante :

```
(DE ASSQ (a al)
    (COND ((NULL al) NIL)
          ((EQ (CAAR al) a) (CAR al))
          (T (ASSQ a (CDR al)))))
```

ex : (ASSQ 'B '((A) (B . 1) (C D E))) ☞ (B . 1)

(CASSQ <a> <al>) [SUBR à 2 argumets]

est identique à ASSQ mais ramène le CDR seul de l'élément sélecté de la A-liste. ATTENTION : on ne peut pas distinguer la valeur NIL d'une variable avec l'absence de cette variable.

ex : (CASSQ 'C '((A) (B . 1) (C D E))) ☞ (D E)

(ASSOC <s> <al>) [SUBR à 2 arguments]

ramène l'élément de la A-liste <al> dont le CAR (la variable) est égal à l'expression <s>, sinon ASSOC ramène NIL. Cette fonction utilise le prédicat EQUAL.

ex : (ASSOC '(X Y) '((A) ((X Y) Z))) ☞ ((X Y) Z)

(CASSOC <s> <al>) [SUBR à 2 arguments]

est identique à ASSOC mais ramène le CDR seul de l'élément sélecté de la A-liste. ATTENTION : on ne peut pas distinguer la valeur NIL d'une variable avec l'absence de cette variable.

ex : (CASSOC '(X Y) '((A) ((X Y) Z))) ☞ (Z)

3.7.2 Création Et Utilisation De A-listes -

(PAIRLIS <l1> <l2> <al>) [SUBR à 3 arguments]

<l1> doit être une liste de variables
<l2> doit être une liste de valeurs
PAIRLIS ramène une nouvelle A-liste formée à partir de la liste des variables et de la liste des valeurs. Si le troisième argument <al> est fourni, il est ajouté à la fin de la A-liste crée.

PAIRLIS peut être défini en VLISP de la manière suivante :

```
(DE PAIRLIS (l1 l2 al)
   (IF (NULL l1)
       al
       (CONS (CONS (CAR l1) (CAR l2))
             (PAIRLIS (CDR l1) (CDR l2) al))))
```

```
ex : (PAIRLIS '(A B C) '(1 2 3)) ☞ ((A . 1)(B . 2)(C . 3))
     (PAIRLIS '(X Y Z) '(5 (6)) '((A . 1) (B . 2)))
       ☞ ((X . 5) (Y 6) (Z) (A . 1) (B . 2))
```

(SUBLIS <al> <s>) [SUBR à 2 arguments]

ramène une copie de l'expression <s> dans laquelle toutes les occurences des variables de la A-liste <al> ont été remplacées par leurs valeurs correspondantes. Cette fonction effectue une copie entière de l'expression <s> et utilise la fonction ASSQ.

SUBLIS peut être défini en VLISP de la manière suivante :

```
(DE SUBLIS (A E)
  (LET ((E E))
       (IF (ATOM E)
           (LET ((X (ASSQ E A)))
                (IF X (CDR X) E))
       (CONS (SELF (NEXTL E)) (SELF E)))))
```

```
ex :  (SUBLIS '((A . 1) (B 2 3)) '(A (B C) D B . B)))
   ☞ (1 ((2 3) C) D (2 3) 2 3)
```

3.8 Fonctions Sur Les P-listes

En VLISP 10 comme dans tous les VLISP, les P-listes (listes de propriétés) sont des listes qui ont la structure suivante :
  (indic1 val1 indic2 val2 ... indicN valN)
A chaque indicateur est associé une valeur. Les recherches sur les P-listes s'effectuent donc deux éléments par deux éléments.

Les arguments de ces fonctions sont :
<pl> - si <pl> est un atome littéral, la P-liste utilisée sera le CDR de l'atome.
  - si <pl> est une liste, la P-liste utilisée sera cette liste elle-même.
  - si <pl> est un nombre ou une chaîne, toutes ces fonctions ramènent la valeur NIL.
<ind> l'indicateur est une expression quelconque, la recherche des indicateurs utilise donc le prédicat EQUAL. Toutefois pour accélérer la recherche, les fonctions GET, GETL, PUT et REMPROP testent d'abord si l'indicateur n'est pas un atome littéral; en ce cas, une routine spéciale [illegible]

employée. Cette routine utilise le prédicat EQP et est extrêmement rapide.
<lind> est une liste d'indicateurs.
<pval> peut être n'importe quelle expression

3.8.1 Fonctions De Recherche Sur P-liste -

(GET <pl> <ind>) [SUBR à 2 arguments]

ramène la valeur associée à l'indicateur <ind> dans la P-liste <pl>. Si l'indicateur n'existe pas, GET ramène NIL.
ATTENTION : on ne peut pas discerner la valeur égale à NIL d'un indicateur avec l'absence de cet indicateur.

GET peut être défini en VLISP de la manière suivante :

```
(DE GET (pl ind)
   ((LET ((pl (COND
                 ((LITATOM pl) (CDR pl))
                 ((LISTP pl) pl)
                 (T NIL))))
      (COND ((NULL pl) NIL)
            ((EQUAL (CAR pl) ind) (CADR pl))
            (T (SELF (CDDR pl))))))
```

ex : (GET '(I1 A I2 B) 'I2) ☞ B

(GETL <pl> <lind>) [SUBR à 2 arguments]

<pl> est la P-liste à explorer et <lind> est une liste d'indicateurs. GETL ramène la sous-P-liste de <pl> commençant par un des éléments de <lind>. Cette fonction permet entre autres de lever l'ambiguité qui existait, dans la fonction GET, entre l'absence d'un indicateur et la valeur NIL associée à un indicateur.

ex : (GETL '(I1 1 I2 2 I3 3) '(I2 I4)) ☞ (I2 2 I3 3)
(GETL '(I1 1 I2 NIL) '(I2)) ☞ (I2 NIL)

3.8.2 Création Et Modification De P-liste -

(ADDPROP <pl> <pval> <ind>) [SUBR à 3 arguments]

rajoute en tête de la P-liste <pl> l'indicateur <ind> et sa valeur associée <pval>. ADDPROP ramène <pl> en valeur.

ex : (ADDPROP '(I1 A I2 B) 'C 'I1) ☞ (I1 C I1 A I2 B)

(PUT <pl> <pval> <ind>) [SUBR à 3 arguments]

si l'indicateur <ind> existe déjà sur la P-liste <pl>, sa valeur associée prend la nouvelle valeur <pval>, sinon l'indicateur <ind> et sa valeur associée <pval> sont ajoutés en tête de <pl> (d'une manière identique à la fonction ADDPROP). PUT ramène <pl> en valeur.

ex : (PUT '(I1 A I2 B) 'C 'I1) ☞ (I1 C I2 B)

(DEFPROP <pl> <pval> <ind>) [FSUBR]

est équivalent à PUT mais aucun argument n'est évalué.

ex : (DEFPROP FOO (LAMBDA () (FUU)) EXPR) ☞ FOO

(REMPROP <pl> <ind>) [SUBR à 2 arguments]

enlève de la P-liste <pl> l'indicateur <ind> s'il existe ainsi que sa valeur associée. REMPROP ramène <pl> en valeur.
L'utilisation conjuguée des fonctions REMPROP et ADDPROP permet d'utiliser les P-listes comme des piles de propriété-valeurs.

ex : (REMPROP '(I1 A I2 B) 'I2) ☞ (I1 A)

3.9 Fonctions Sur Les P-names Des Atomes

Rappelons que :
- le P-Name d'un atome littéral est son nom externe
- le P-Name d'un nombre est sa représentation externe dans la base de conversion de sortie courante.
- le P-Name d'une chaîne est la suite de caractères qui constituent cette chaîne.

(PLENGTH <a>) [SUBR à 1 argument] {pour Pname LENGTH}

ramène le nombre de caractères du P-Name de l'atome <a>. (PLENGTH <a>) correspond à (LENGTH (EXPLODE <a>)).

```
ex : (PLENGTH  -128)      ☞ 4
     (PLENGTH  "A")       ☞ 3
     (PLENGTH 'TOPLEVEL)  ☞ 8
```

(SORT <a1> <a2>) [SUBR à 2 arguments]

ramène T si le P-Name de <a1> est inférieur ou égal (lexicographiquement) au P-name de <a2>, sinon ramène NIL. Cette fonction est utilisée pour réaliser des tris alphabétiques.

```
ex : (SORT 'A 'A)       ☞ T
     (SORT 'B 'A)       ☞ NIL
     (SORT 'A 'B)       ☞ T
     (SORT 'ZZZ 'ZZZZ)  ☞ T
```

(SAMEPN <a1> <a2>) [SUBR à 2 arguments] {pour SAME P-Name}

ramène T si le P-Name de l'atome <a1> commence par le P-Name de l'atome <a2>, sinon ce prédicat ramène NIL. SAMEPN est utilisé pour tester les premiers caractères du P-Name d'un atome.

```
ex :  (SAMEPN '*L101 '* )   ☞ T
      (SAMEPN '*L101 '*L)   ☞ T
      (SAMEPN '*L101 '*M)   ☞ NIL
```

(GENSYM <a1> ... <aN>) [SUBR à N arguments] {pour GENerate SYMbol}

si aucun argument n'est donné, GENSYM ramène à chaque appel un nouvel atome littéral de type Gxxx dans lequel xxx est un nombre incrémenté à chaque appel; xxx vaut 100 au départ de l'interprète.
Si des arguments sont fournis, GENSYM ramène un nouvel atome dont le P-name est la concaténation de tous les P-names des arguments atomiques <a1> ... <aN> jusqu'à concurrence de 13 caractères.

```
ex : (GENSYM)                     ☞ G100
     (GENSYM)                     ☞ G101
     (GENSYM)                     ☞ G102
     (GENSYM 'LAB (ADD1 3) ':)    ☞ LAB4:
```

(EXPLODE <a1> ... <aN>) [SUBR à N arguments]

ramène la liste concaténée de tous les caractères des P-names des arguments atomiques <a1> ... <aN>.

```
ex : (EXPLODE 'NUM -237 'HAA)  ☞  (N U M - 2 3 7 H A A)
```

(ASCII <n>) [SUBR à 1 argument]

ramène le caractère de code ASCII <n> (modulo 256).

```
ex : (ASCII 67)          ☞ C
     (ADD1 (ASCII 49))   ☞ 2
```

(CASCII <c>) [SUBR à 1 argument] {pour Code ASCII}

ramène le code ASCII du caractère <c>. Un caractère étant un atome dont le PLENGTH est égal à 1.

```
ex : (CASCII 'C)   ☞ 67
     (CASCII 1)    ☞ 49
```

3.10 Les Fonctions STATUS

Les fonctions magiques STATUS permettent tout ce que vous avez toujours eu envie de faire à un système LISP sans jamais pouvoir le faire vraiment. D'une manière plus précise, les fonctions STATUS permettent de connaître et/ou de modifier les valeurs des indicateurs et des variables internes du système VLISP.

(STATUS <n> <arg1> ... <argN>) [SUBR à N arguments]

Le 1er argument <n> est le numéro de la fonction STATUS à exécuter. Dans l'état actuel du système une trentaine de fonctions sont disponibles. Un numéro de STATUS incorrect déclenche une erreur avec impression du libellé :
** STATUS error : <n>.
dans lequel <n> est le numéro de la fonction STATUS incorrect.

Hormis celles manipulant directement le R.G., la plupart des fonctions STATUS sont décrites dans les autres chapitres.

3.10.1 Le Registre Général (R.G.) -

Il existe un registre de 36 bits, le registre général R.G. , qui contient les indicateurs principaux du système.
La signification de chacun de ces bits est mentionnée dans les autres chapitres, voici simplement un tableau récapitulatif de la signification des différents bits du R.G.

no bit	val/ def.	si le bit est à 1
0	1	imprime le temps d'une évaluation au top-level (cf: la fonction TOPLEVEL).
1	1	imprime le résultat de toutes les évaluations au top-level (cf: la fonction TOPLEVEL).
2	1	imprime les formes à évaluer au top-level (cf: la fonction TOPLEVEL).
3	0	trace tous les appels internes de la fonction EVAL (cf: le chapitre erreurs et mise-au-point).
4	0	trace tous les appels internes de la fonction APPLY (cf: le chapitre erreurs et mise-au-point).
5	0	édite des statistiques après chaque G.C. (cf: le Garbage collecting).
6	0	édite une 3ème WHO line après chaque G.C. (cf: le Garbage collecting).
7	1	teste la validité de l'indice des tableaux. (cf: les tableaux).
8	0	trace tous les appels internes de la fonction EVAL en mode pas-à-pas (cf: les fonctions STEP et UNSTEP)
9	0	non utilisé actuellement
10	0	imprime l'image de tous les enregistrements lus en entrée. (cf: les fonctions d'entrée)
11	1/0	le fichier d'entrée est un fichier TTY. Il y a donc impression du caractère ? avant chaque lecture de ligne et activation d'un petit pretty-print. Ce bit est positionné automatiquement par la fonction INPUT.
12	0	en entrée les nombres peuvent commencer par le signe + (cf: les nombres).
13	1	en entrée les nombres peuvent commencer par le signe - (cf: les nombres).
14	1	en entrée l'action du quote-caractère (par défaut /) est validée.
15	1	en entrée les macro-caractères sont traités.
16	1	en entrée les macro-fonctions sont traitées.

17	1	en entrée les chaînes de caractères sont acceptées.
18	1	en entrée les commentaires sont acceptés.
19	1	en entrée les caractères minuscules sont transcodés automatiquement en caractères majuscules.
20	1	les impressions physiques ont lieu à la fin de chaque ligne (TERPRI) et non à la fin du buffer de sortie. (cf: les fonctions de sortie).
21	1	en sortie toute édition est précédée d'un espace.
22	0	en sortie les nombres positifs commencent par le signe +
23	1	en sortie les nombres négatifs commencent par le signe -
24	0	en sortie les caractères spéciaux des P-names sont précédés du caractère spécial / .
25	1	en sortie il y a un espace entre chaque atome.
26	1	en sortie les macro-fonctions sont traitées.
27	1	en sortie le caractère délimiteur de chaîne est restitué.
28	1	le préfixe du buffer de sortie est imprimé.
29	1/0	l'interprète a été lancé par l'éditeur ETV.
30	1/0	le système se trouve dans la fonction IMPLODE.
31	1/0	le système se trouve dans la fonction LIBRARY.
32	1/0	le système se trouve dans la fonction READ.
33	1/0	il y a eu une interruption de type ESCAPE.I.
34	1/0	le système se trouve en début de démarrage à chaud.
35	1/0	le système se trouve dans un G.C.

ces bits sont manipulés par les fonctions STATUS suivantes :

(STATUS 0 <n>)

si <n> est un nombre, donne cette nouvelle valeur au R.G. Ce STATUS ramène le nombre qui représente la valeur du R.G. courant.

(STATUS 1 <n1> ... <nN>)

met à 1 les bits du R.G. de numéro <n1> ... <nN>. Si un de ces arguments n'est pas un nombre compris entre 0 et 35, l'erreur status se déclenche. Ce status ramène <nN> en valeur.

(STATUS 2 <n1> ... <nN>)

est identique à (STATUS 1 ...) mais les bits sélectionnés sont remis à 0.

(STATUS 3 <n1> ... <nN>)

est identique à (STATUS 1 ...) mais les bits sélectionnés sont inversés.

(STATUS 4 <n1> ... <nN>)

teste les différents bits <n1> ... <nN> du R.G. Si tous ces bits sont positionnés à 1, ce status ramène <nN> sinon ce status ramène NIL.

3.11 Les Fonctions Système

(DATE) [SUBR à 0 argument]

ramène un atome représentant la date du jour sous la forme :
 jj-mmm-aa
mmm sont les trois premières lettres du mois (en anglais).

ex : (DATE) ☞ 15-Aug-78

(TIME) [SUBR à 0 argument]

ramène un atome représentant l'heure courante sous la forme :
 hh:mm:ss
TIME ramène un atome littéral prêt à être édité. On ne peut pas utiliser cet atome pour effectuer des calculs de durée (pour cela il est nécessaire d'utiliser la fonction DAYTIME).

ex : (TIME) ☞ 12:27:45
 (TIME) ☞ 12:27:52

(RUNTIME) [SUBR à 0 argument]

ramène le temps d'unité centrale (en milli-secondes) utilisé depuis le LOGIN. Cette fonction est utilisée pour calculer le temps machine consommé par l'interprète.

(DAYTIME) [SUBR à 0 argument]

ramène l'heure du jour en milli-secondes depuis minuit. Cette fonction est utilisée pour calculer des durées réelles.

(SWITCH) [SUBR à 0 argument]

ramène un nombre représentant la valeur affichée aux clés du pupitre du PDP10.

(LIGHTS <n>) [SUBR à 1 argument]

affiche sur les voyants du pupitre du PDP10 la valeur de l'argument numérique <n>.

(PJOB) [SUBR à 0 argument]

ramène le numéro de votre JOB. Cette fonction utilise le PJOB UUO et correspond à la commande moniteur : .PJOB

(CALLI <n1> <n2>) [SUBR à 2 arguments]

permet d'utiliser l'UUO CALLI d'une manière standard (la description de cette UUO se trouve dans la documentation DEC) <n1> est le numéro du CALLI désiré, <n2> est le numéro du registre à employer. Cette fonction n'est utilisée que dans certaines applications système trés spécialisées.

(GETTAB <n1> <n2>) [SUBR à 2 arguments]

ramène la valeur du GETTAB UUO (qui donne accès aux tables du moniteur) avec <n1> comme index et <n2> comme numéro de table. De la même manière que CALLI, cette fonction n'est utilisée que dans certaines applications système trés spécialisées.

(VERSION) [SUBR à 0 argument]

ramène le numéro de version de l'interprète utilisé. Ce numéro se code en binaire :

```
lll jjjjjjjjj kkkkkk llllllllllllllllll
 3      9        6           18
```

dans lequel :

jjj... = est le numéro de version majeure (e.g. 10)
kkk... = est le numéro de version mineure (e.g. 3)
lll... = est le numéro d'édition (e.g. 22)

cette fonction est utilisée dans les programmes système de type LODLAP ou COMPIL pour s'assurer de la compatibilité des interprètes dans le temps. Pour être plus éclairé, évaluez plutôt l'atome VERSION.

(IRCAMP) [SUBR à 0 argument]

ce prédicat ramène T si l'interpréte possède les modifications pour être exécuté sur le système de l'IRCAM. Ces modifications portent principalement sur les terminaux de visualisation, sur certaines UUO systèmes de manipulation de fichiers (SHOWIT ...), sur les noms des répertoires (ALIAS, ppn ...). Si l'interpréte est prévu pour utiliser le système TOPS 10 classique, IRCAMP ramène NIL.

(RESET <i>) [SUBR à 1 argument]

arrête l'évaluation en cours, et retourne directement au TOPLEVEL. La place occupée par les buffers d'entrée sortie système est libérée, le messsage RESET est imprimé sur le terminal, enfin, les fichiers d'entrée sortie standard sont ouverts.
Si l'indicateur <i> = NIL, toutes les variables gardent les valeurs

qu'elles possèdaient au moment du RESET (elles ne sont donc pas déliées); en revanche si l'indicateur <i> = T, toutes les variables sont déliées et reprennent les valeurs qu'elles possèdaient au niveau du top-level.

exemple d'utilisation du RESET

```
?
?  (SETQ X 1)
=  1
?
?  (LET ((X 10))
?     (PRINT X)
?     (RESET))
   10
RESET

  ---  ALLO ?  ---
?
?  X
=  10
?
?  (LET ((X 20))
?     (PRINT X)
?     (RESET T))
   20
RESET

  ---  ALLO ?  ---
?
?  X
=  10
```

(STOP) [SUBR à 0 argument]

arrête l'évaluation en cours, ferme tous les fichiers ouverts, sort de l'interprète, ne passe pas par la case départ et rend le contrôle au moniteur.
VLISP sentant sa fin venir imprime sur le terminal : Bye et exécute le EXIT UUO qui, lui, imprime sur le terminal : EXIT. Cette fonction permet de sortir de l'interprète de la facon la plus naturelle. Arrêter l'interprète au moyen d'un ↑C risque de faire perdre les fichiers disques ouverts à cet instant. Il existe un macro-caractère définie dans SYS:VLISP.INI (cf: appendice A) qui réalise cette fonction, le caractère "flèche gauche" ou "blanc souligné" :

```
?  (DMC /← () (STOP))
=  ←
?  ←
Bye
EXIT
.
```



CHAPITRE 4

LES NOMBRES

VLISP possède deux types de nombres : les nombres entiers et les nombres flottants. Ils sont stockés dans une zone de longeur fixe gérée dynamiquement. Si cette zone se révèle trop petite, le message d'erreur suivant apparait :
** no room for numbers.
Cette erreur est fatale. Il faut augmenter la taille de la zone allouée aux nombres (dans la fonction CONFIGURATION du fichier initial CONFIG.INI) et relancer le travail.

Si durant un calcul, une exception arithmétique se produit, une erreur apparait, avec comme libellé :
** arithmetic exception. PC : <pc>
dans lequel <pc> est la valeur du Program Control. Ce PC contient en partie droite la valeur du compteur ordinal de l'instruction qui a provoqué l'interruption et en partie gauche l'état des indicateurs (en particulier les indicateurs débordement d'entier , débordement flottant , division par 0 ... cf: SYSTEM REF. MANUAL pp. 2-58 pour avoir une description complète de ces indicateurs).

Certaines fonctions (les fonctions de conversion, les fonctions de l'arithmétique mixte et les fonctions de comparaison mixte) testent si leur(s) argument(s) sont des nombres. Dans le cas où ils ne le seraient pas, une erreur apparait. Le libellé de cette erreur est :
<fonction> : ** non numeric arg : <argument>
dans lequel le nom de la fonction et l'argument incriminé sont imprimés.

Un certain nombre de fonctions mathématiques ont été empruntées à la bibliothèque FORTRAN (ex: les fonctions SQRT, SIN, ATAN ...). Ces fonctions émetttent parfois des diagnostics d'erreurs qui leurs sont propres (par ex: la tentative de calcul de la racine carré d'un nombre négatif ...). Ces erreurs ne sont pas fatales sous VLISP mais le résultat du calcul est bien évidemment erronné.

4.1 Représentation Des Nombres

La représentation externe des nombres entiers est une suite de chiffres dans la base de numération courante. Cette base est définie par les fonctions :
(STATUS 5 <n>) en entrée
(STATUS 6 <n>) en sortie
dans lesquelles <n> est un nombre compris entre 2 et 16. Par défaut, la valeur de ces bases est 10.

Il existe un macro-caractère standard qui permet de lire des S-expressions en supposant que tous les nombres qui y sont inclus sont représentés en octal, l'anti-slash \ . Il possède la définition suivante :

```
(DMC "\" ()
   (PUSH (STATUS 5))       ; sauve l'ancienne base ;
   (STATUS 5 8)            ; base d'entrée octale ;
   (PROG1                  ; puis lit une S-expression (qui ;
      (READ)               ; sera ramenée en valeur) et ;
      (STATUS 5 (POP))))   ; restaure l'ancienne base ;
```

ex : \(730 . 272) est équivalent à (472 . 186)

Les nombres entiers peuvent être signés ou non. Le traitement des signes est controlé par des bits du r.g. :
bit 12 du r.g. en entrée un nombre peut débuter par le signe +
bit 13 du r.g. en entrée un nombre peut débuter par le signe -
bit 22 du r.g. en sortie les nombres positifs débutent par le signe +
bit 23 du r.g. en sortie les nombres négatifs débutent par le signe -

Ces nombres sont stockés en mémoire sur un mot (de 36 bits); ils doivent donc être compris dans l'intervalle [-2 ↑ 35 , +2 ↑ 35 - 1] c'est-à-dire [-34359738368 , +34359738367].

La représentation externe des nombres flottants est une suite de chiffres décimaux suivie immédiatement par un point décimal (.) et suivie optionnellement par une fraction décimale également. Pour garder une précision de 8 digits un exposant peut être rajouté en notation "E". Cette dernière forme d'écriture n'est toutefois disponible qu'en sortie actuellement. Les nombres flottants sont stockés en mémoire sur un mot (de 36 bits) qui contient 1 bit de signe, 8 bits d'exposant et 27 bits de mantisse.

Il n'existe à l'heure actuelle ni de nombres flottants double précision, ni de nombres complexes, ni de Bignums.

4.2 Les Tests De Type

(NUMBP <s>) [SUBR à 1 argument] {pour NUMBer Predicate}

Cette fonction sert à tester si l'argument <s> est un nombre de n'importe quel type. si l'argument <s> est un nombre, NUMBP ramène <s> sinon NUMBP ramène NIL.

```
ex : (NUMBP (ADD1 67))    ☞ 68
     (NUMBP -56.89)       ☞ -56.89
     (NUMBP "Nan")        ☞ NIL
```

(INUMBP <s>) [SUBR à 1 argument]

teste si l'argument <s> est un "petit entier" i.e. un nombre entier ayant une représentation unique dans l'interprète. Si <s> est un petit entier , INUMBP ramène ce nombre sinon INUMBP ramène NIL. Cette fonction est principalement utilisée par le compilateur pour améliorer le code généré : en effet les comparaisons de petits entiers se font sur des adresses et sont donc très rapide. Le nombre de petits entiers est déterminé au moment de la génération du système VLISP. D'une manière standard tous les nombres entiers dans l'intervalle [-127 , +511] sont des petits entiers.

```
ex : (INUMBP 78)      ☞ 78
     (INUMBP -2558)   ☞ NIL
     (EQP 12 12)      ☞ T (toujours, si 12 est un petit entier)
```

(FIXP <s>) [SUBR à 1 argument] {pour FIX Predicate}

Cette fonction sert à tester si l'argument <s> est un nombre entier. Si <s> est un nombre entier FIXP ramène ce nombre sinon FIXP ramène NIL.

```
ex : (FIXP 78)      ☞ 78
     (FIXP -8878)   ☞ -8878
     (FIXP 12.2)    ☞ NIL
```

(FLOATP <s>) [SUBR à 1 argument] {pour FLOAT Predicate}

Cette fonction sert à tester si l'argument <s> est un nombre flottant. Si <s> est un nombre flottant FLOATP ramène ce nombre sinon FLOATP ramène NIL.

```
ex : (FLOATP 100)     ☞ NIL
     (FLOATP 100.1)   ☞ 100.1
```

(TYPNUMB <s>) [SUBR à 1 argument] {pour TYPe NUMBer}

cette fonction teste le type du nombre donné en argument et ramène l'atome :
- FIX si l'argument <n> est de type entier,
- FLOAT si l'argument <n> est de type flottant,
- NIL si l'argument <n> n'est pas un nombre.

```
ex : (TYPNUMB 14)     ☞ FIX
     (TYPNUMB 14.4)   ☞ FLOAT
     (TYPNUMB "ou")   ☞ NIL
```

4.3 Les Conversions

Ces fonctions testent si leur argument est un nombre. Dans le cas où il ne l'est pas, une erreur apparait. Le libellé de cette erreur est :
<fonction> : ** non numeric arg : <argument>
dans lequel le nom de la fonction et l'argument incriminé sont imprimés.

(FIX <n>) [SUBR à 1 argument]

convertit le nombre flottant <n> en son équivalent entier. Un calcul d'arrondi est effectué automatiquement de la manière suivante : la partie entière est incrémentée de 1 si la partie fractionnaire est >= 0.5 (pour les nombres négatifs le test est <= à 0.5).

```
EX : (FIX 3.14)   ☞ 3
     (FIX 3.98)   ☞ 4
     (FIX 3)      ☞ 3
```

(FLOAT <n>) [SUBR à 1 argument]

convertit le nombre entier <n> en son équivalent flottant.

```
ex : (FLOAT 3)           ☞ 3.
     (FLOAT -2)          ☞ -2.
     (FLOAT 120000000)   ☞ 1.2E8
     (FLOAT 3.14)        ☞ 3.14
```

4.4 Arithmétique Entière

Les fonctions qui vont être décrites utilisent des opérandes supposés de type entier. Ces fonctions n'effectuent aucun contrôle de validité de type. Si les arguments ne sont pas des nombres entiers, ces fonctions livrent en général de bien étranges résultats. Toutefois un calcul impossible provoque l'erreur ** arithmetic exception (voir cette erreur).

(ABS <n>) [SUBR à 1 argument]

ramène la valeur absolue de l'argument <n>.

ex : (ABS 10) ☞ 10
(ABS -10) ☞ 10

(ADD1 <n>) [SUBR à 1 argument]

ramène la valeur : <n> + 1 .

ex : (ADD1 6) ☞ 7
(ADD1 -4) ☞ -3

(DIFFER <n1> ... <nN>) [SUBR à N arguments]

ramène la valeur : <n1> - <n2> - ... - <nN>

ex : (DIFFER 6) ☞ 6
(DIFFER 12 7 3) ☞ 2

(MAX <n1> ... <nN>) [SUBR à N arguments]

ramène la valeur maximum des <n1> ... <nN>.

ex : (MAX 1 3 5 3 1) ☞ 5

(MIN <n1> ... <nN>) [SUBR à N arguments]

ramène la valeur minimum des <n1> ... <nN>.

ex : (MIN 5 4 3 4) ☞ 3

(MINUS <n>) [SUBR à 1 argument]

ramène la valeur : - <n>

ex : (MINUS -10) ☞ 10
(MINUS 25) ☞ -25

(PLUS <n1> ... <nN>) [SUBR à N arguments]

ramène la valeur : <n1> + <n2> + ... + <nN>

ex : (PLUS 5 6) ☞ 11
(PLUS 5 6 4) ☞ 15

(QUO <n1> ... <nN>) [SUBR à N arguments]

ramène la valeur du quotient de : <n1> / <n2> / ... / <nN>

ex : (QUO 20 5) ☞ 4
(QUO 40 4 2) ☞ 5

(REM <n1> <n2>) [SUBR à 2 arguments]

ramène la valeur du reste de la division entière de <n1> par <n2>.

ex : (REM 11 3) ☞ 2
(REM 14 22) ☞ 14

(SUB1 <n>) [SUBR à 1 argument]

ramène la valeur : <n> - 1

ex : (SUB1 7) ☞ 6
(SUB1 -9) ☞ -10

(TIMES <n1> ... <nN>) [SUBR à N arguments]

ramène la valeur : <n1> * <n2> * ... * <nN>

ex : (TIMES 10 4) ☞ 40
(TIMES 2 3 4) ☞ 24

4.5 Comparaisons Entieres

Ces fonctions n'effectuent aucun test de validité de type. Si les arguments de ces fonctions ne sont pas des nombres, leurs résultats ne sont pas significatifs. Elles ne provoquent jamais d'erreur.

(EQN <n1> <n2>) [SUBR à 2 arguments] {pour EQ Number}

ramène <n1> si les 2 nombres <n1> et <n2> sont égaux, sinon EQN ramène NIL.

ex : (EQN 58 58) ☞ 58
(EQN 34 34.) ☞ NIL
(EQN 21963283741 21963283741.) ☞ 21963283741
(et c'est bien le seul cas d'égalité mixte!)

(EVENP <n>) [SUBR à 1 argument]

ramène <n> si <n> est pair. Si <n> est impair EVENP ramène NIL.

ex : (EVENP 4) ☞ 4
(EVENP -5) ☞ NIL

(GE <n1> <n2> ... <nN-1> <nN>) [SUBR à N arguments]

si <n1> >= <n2> >= ... >= <nN-1> >= <nN> alors GE ramène <nN-1> sinon GE ramène NIL.

```
ex : (GE 3 7)       ☞ NIL
     (GE 7 7 4 3)   ☞ 4
```

(GEZP <n>) [SUBR à 1 argument]

ramène <n> si <n> est plus grand ou égal à 0. Si <n> est plus petit que 0 GEZP ramène NIL.

```
ex : (GEZP 5)    ☞ 5
     (GEZP 0)    ☞ 0
     (GEZP -5)   ☞ NIL
```

(GT <n1> <n2> ... <nN-1> <nN>) [SUBR à N arguments]

si <n1> > <n2> > ... > <nN-1> > <nN> alors GT ramène <nN-1> sinon GT ramène NIL.

```
ex : (GT 5 5)     ☞ NIL
     (GT 7 4 3)   ☞ 4
```

(GZP <n>) [SUBR à 1 argument]

ramène <n> si <n> est plus grand que 0. Si <n> est plus petit ou égal à 0 GZP ramène NIL.

```
ex : (GZP 5)    ☞ 5
     (GZP 0)    ☞ NIL
     (GZP -5)   ☞ NIL
```

(LE <n1> <n2> ... <nN-1> <nN>) [SUBR à N arguments]

si <n1> <= <n2> <= ... <= <nN-1> <= <nN> alors LE ramène <nN-1> sinon LE ramène NIL.

```
ex : (LE 5 5)       ☞ 5
     (LE 4 4 6 9)   ☞ 9
```

(LEZP <n>) [SUBR à 1 argument]

ramène <n> si <n> est plus petit ou égal à 0. Si <n> est plus grand que 0 LEZP ramène NIL.

```
ex : (LEZP 5)    ☞ NIL
     (LEZP 0)    ☞ 0
     (LEZP -5)   ☞ -5
```

(LT <n1> <n2> ... <nN-1> <nN>) [SUBR à N arguments]

si <n1> < <n2> < ... < <nN-1> < <nN> alors LT ramène <nN-1> sinon LT ramène NIL.

```
ex : (LT 5 5)     ☞ NIL
```

(LT 4 5 6) ☞ 5

(LZP <n>) [SUBR à 1 argument]

ramène <n> si <n> est plus petit que 0. Si <n> est plus grand ou égal à 0 LZP ramène NIL.

```
ex : (LZP 5)   ☞ NIL
     (LZP 0)   ☞ NIL
     (LZP -5)  ☞ -5
```

(NEQN <n1> <n2>) [SUBR à 2 arguments] {pour Not EQ Number}

ramène <n1> si les deux nombres <n1> et <n2> sont différents. NEQN ramène NIL si les deux nombres sont égaux. Cette fonction est équivalente à (NOT (EQN <n1> <n2>)).

```
ex : (NEQN 5 6)    ☞ 5
     (NEQN -3 -3)  ☞ NIL
```

(NEROP <n>) [SUBR à 1 argument]

ramène <n> si <n> est différent de 0. Si <n> égal 0 NEROP ramène NIL.

```
ex : (NEROP 5)   ☞ 5
     (NEROP 0)   ☞ NIL
     (NEROP -5)  ☞ -5
```

(ODDP <n>) [SUBR à 1 argument]

ramène <n> si <n> est impair. Si <n> est pair ODDP ramène NIL.

```
ex : (ODDP -4)  ☞ NIL
     (ODDP -5)  ☞ -5
```

(ZEROP <n>) [SUBR à 1 argument]

ramène 0 si <n> est égal à 0 sinon ramène NIL.

```
ex : (ZEROP 5)   ☞ NIL
     (ZEROP 0)   ☞ 0
     (ZEROP -5)  ☞ NIL
```

4.6 Arithmétique Mixte

Ces fonctions testent si leurs arguments sont des nombres. Dans le cas où ils ne le sont pas, une erreur apparait. Le libellé de cette erreur est :

```
<fonction> : ** non numeric arg : <argument>
```

dans lequel le nom de la fonction et l'argument incriminé sont imprimés.

Ces arguments peuvent être de n'importe quel type : entier ou flottant. Si l'un des arguments est flottant, le résultat est un nombre flottant ; si tous les arguments sont de type entier, le résultat est un nombre entier. Il y a donc conversion automatique des arguments avec priorité aux nombres flottants.

(1+ <n>) [SUBR à 1 argument]

ramène la valeur : <n> + 1 .

ex : (1+ 3) ☞ 4
(1+ 1.2) ☞ 2.2

(1- <n>) [SUBR à 1 argument]

ramène la valeur : <n> - 1 .

ex : (1- 3) ☞ 2
(1- 3.2) ☞ 2.2

(+ <n1> <n2>) [SUBR à 2 arguments]

ramène la somme des 2 arguments : <n1> + <n2> .

ex : (+ 2 3) ☞ 5
(+ 2 3.2) ☞ 5.2
(+ 2.3 3) ☞ 5.3
(+ 2.3 3.3) ☞ 5.6

(- <n1> <n2>) [SUBR à 2 arguments]

ramène la différence des 2 arguments : <n1> - <n2> .

ex : (- 3 1) ☞ 2
(- 3 1.) ☞ 2.
(- 3.2 1.1) ☞ 2.1

(* <n1> <n2>) [SUBR à 2 arguments]

ramène le produit des 2 arguments : <n1> * <n2> .

ex : (* 3 2) ☞ 6
(* 3 2.1) ☞ 6.3
(* 3. 2.2) ☞ 6.6

(// <n1> <n2>) [SUBR à 2 arguments]

ramène le quotient des 2 arguments : <n1> // <n2> . Le caractère / est un caractère spécial, il faut le doubler pour utiliser cette fonction : on doit écrire (// 3 2.) et non (/ 3 2.).

ex : (// 3 2) ☞ 1
(// 3 2.) ☞ 1.5
(// 1 3.) ☞ 0.3333333

(** <n1> <n2>) [SUBR à 2 arguments]

ramène la valeur de l'évaluation de <n1> à la puissance <n2>.

ex : (** 2 3) ☞ 8
(** 2. 4) ☞ 16.
(** 2 5.) ☞ 32.
(** 2 0.5) ☞ 1.414214

(** 2 -0.5) ☞ 0.7071068
(** 15. 0) ☞ 1.

(/\ <n1> <n2>) [SUBR à 2 arguments]

ramène le reste de la division de <n1> et <n2>. Cette fonction est utilisée en général avec des arguments entiers pour ramener la valeur du reste d'une division entière (la fonction MODULO). Le caractère \ étant un macro-caractère, il faut le faire précéder du caractère / (quote caractère) pour utiliser cette fonction. On doit écrire (/\ 5 3) et non (\ 5 3) .

ex : (/\ 5 3) ☞ 2
(/\ 5. 3.) ☞ 2.980232E-8

4.7 Comparaisons Mixtes

Ces fonctions testent si leurs deux arguments sont des nombres. Dans le cas où ils ne le seraient pas, une erreur apparait. Le libellé de cette erreur est :
<fonction> ** non numeric arg : <argument>
dans lequel le nom de la fonction et l'argument incriminé sont imprimés.

Si les deux arguments sont de type entier, ces fonctions effectuent des comparaisons entières (elles sont donc équivalentes aux fonctions de comparaison entières), si au moins un des arguments est flottant, les comparaisons s'opèrent en flottant après conversion, si nécessaire de l'autre argument.

(= <n1> <n2>) [SUBR à 2 arguments]

ramène <n1> si <n1> et <n2> sont égaux. Si <n1> est différent de <n2>, = ramène NIL.

ex : (= 378 378.) ☞ 378.
(= 22 23) ☞ NIL

(# <n1> <n2>) [SUBR à 2 arguments]

ramène <n1> si <n1> est différent de <n2>. Si <n1> est égal à <n2>, # ramène NIL.

ex : (# 327.12 312) ☞ 327.12
(# 22.0 22) ☞ NIL

(> <n1> <n2>) [SUBR à 2 arguments]

ramène <n1> si <n1> est supérieur à <n2>. Si <n1> est inférieur ou égal à <n2>, > ramène NIL.

ex : (> 327.12 327) ☞ 327.12
(> 327 327.1) ☞ NIL

(>= <n1> <n2>) [SUBR à 2 arguments]

ramène <n1> si <n1> est supérieur ou égal à <n2>. Si <n1> est inférieur à <n2>, >= ramène NIL.

```
ex : (>= 327. 327)     ☞  327.
     (>= 327 327.1)    ☞  NIL
```

(<= <n1> <n2>) [SUBR à 2 arguments]

ramène <n1> si <n1> est inférieur ou égal à <n2>. Si <n1> est supérieur à <n2>, <= ramène NIL.

```
ex : (<= 327. 327)     ☞  327.
     (<= 327.1 327)    ☞  NIL
```

(< <n1> <n2>) [SUBR à 2 arguments]

ramène <n1> si <n1> est inférieur à <n2>. Si <n1> est supérieur ou égal à <n2>, < ramène NIL.

```
ex : (< 327.12 327)  ☞  NIL
     (< 327 327.1)   ☞  327
```

4.8 Fonctions Logiques

Pour toutes les fonctions qui vont être décrites, l'argument s'il existe doit être de type entier. Il n'y a pas de conversion automatique. Ces fonctions ne provoquent jamais d'erreur.

(COMPL <n>) [SUBR à 1 argument]

ramène la valeur du complément logique de <n>

```
ex : (COMPL 0)    ☞  -1
     (COMPL -2)   ☞  1
```

(LOGAND <n1> <n2>) [SUBR à 2 arguments] {pour LOGical AND}

Effectue l'opération de ET logique entre les deux opérandes <n1> et <n2>. (LOGAND <n1> <n2>) est équivalent à l'ancienne écriture : (BOOLE 5 <n1> <n2>).

```
ex : (LOGAND \36 \25)    ☞  \24
     pour savoir si <n> est une puissance de 2 :
     (EQN <n> (LOGAND <n> (MINUS <n>)))
```

(LOGSHIFT <n1> <n2>) [SUBR à 2 arguments] {pour LOGical SHIFT}

Effectue un décalage logique de la valeur <n1>, <n2> fois. Si <n2> est >0 un décalage gauche s'effectue, dans le cas contraire (<n2> < 0), le décalage s'effectue à gauche.

```
ex : (LOGSHIFT 8 3)    ☞  64
     (LOGSHIFT 8 -2)   ☞  2
```

(LOGOR <n1> <n2>) [SUBR à 2 arguments] {pour LOGical OR}

effectue l'opération de OU logique entre les deux opérandes <n1> et <n2>. (LOGOR <n1> <n2>) est équivalent à l'ancienne écriture : (BOOLE 1 <n1> <n2>).

ex : (LOGOR \15 \7) ☞ \17

(LOGXOR <n1> <n2>) [SUBR à 2 arguments] {pour LOGical XOR}

effectue l'opération de OU exclusif logique entre les deux opérandes <n1> et <n2>. (LOGXOR <n1> <n2>) est équivalent à l'ancienne écriture : (BOOLE 9 <n1> <n2>).

ex : (LOGXOR 5 3) ☞ 6

(SWAP <n>) [SUBR à 1 argument]

ramène la valeur résultant de l'échange des 18 bits de droite avec les 18 bits de gauche de <n>.

ex: (SWAP \177333001) ☞ \333001000177

4.9 Fonctions Mathématiques

Pour toutes les fonctions qui vont être décrites, l'argument s'il existe doit être de type flottant. Il n'y a pas de conversion automatique.

(SQRT <n>) [SUBR à 1 argument]

ramène la racine carrée du nombre <n>. <n> doit être un nombre flottant positif.

```
ex : (SQRT 25.)  ☞ 5.
     (SQRT 5.)   ☞ 2.236068
     (SQRT -5.)  ☞
%FRSLIB attempt to take SQRT of negative arg= 2.236068
```

Ce message d'avertissement est donné par la fonction en cas d'argument négatif.

(SIN <n>) [SUBR à 1 argument]

ramène la valeur du sinus de l'angle <n> exprimé en radians.

(COS <n>) [SUBR à 1 argument]

ramène la valeur du cosinus de l'angle <n> exprimé en radians.

(ATAN <n>) [SUBR à 1 argument]

ramène la valeur de la fonction arc tangente, c'est-à-dire la valeur de l'angle (en radians) dont la tangente est <n>.

; exemples d'utilisation des fonctions trigonométriques ;

```
(SETQ PI    3.14159)   ☞  3.14159
(SETQ PI:2 (// PI 2))  ☞  1.570795
(SETQ PI:4 (// PI 4))  ☞  0.7853975

(SIN 0.)          ☞  0
(SIN PI:4)        ☞  0.7071063
(SIN PI:2)        ☞  1.

(COS 0.)          ☞  1.
(COS PI:4)        ☞  0.7071072
(COS PI:2)        ☞  1.334181E-6

(SETQ X 1.2)           ☞  1.2

(SQRT (+ (** (SIN X) 2) (** (COS X) 2)))  ☞  1.

; la fonction TANGENTE n'est pas standard : définissons-la ;

(DE TANG (X)
   (// (SIN X) (COS X)))  ☞  TANG

(TANG 0.)              ☞  0
(TANG PI:4)            ☞  0.9999987
(TANG PI:2)            ☞  749523.3

(ATAN 0.)              ☞  0
(ATAN 1.)              ☞  0.7853982
(ATAN (TANG 1.123))    ☞  1.123
```

(EXP <n>) [SUBR à 1 argument]

ramène la valeur e puissance <n>.

```
ex : (EXP 0)      ☞  1.
     (EXP 1.)     ☞  2.718282
```

(LOG <n>) [SUBR à 1 argument]

ramène la valeur du logarithme **népérien de** <n>.

```
ex : (LOG 1.)          ☞  0
     (LOG 2.718282)    ☞  1.
```

(LOG10 <n>) [SUBR à 1 argument]

ramène la valeur du logarithme **décimal de** <n>.

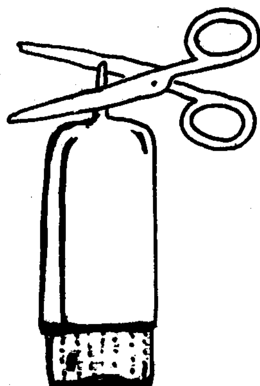
```
ex : (LOG10 1.)        ☞ 0
     (LOG10 10.)       ☞ 1.
     (LOG10 20.)       ☞ 1.30103
     (LOG10 100.)      ☞ 2.
```

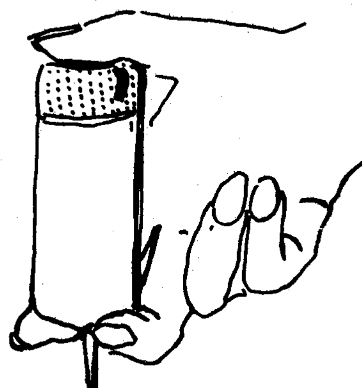
(RANDOM) [SUBR à 0 argument]

ramène à chaque appel un nombre **flottant, aléatoire, compris** dans l'intervalle]0. , 1.[.

```
ex : (RANDOM)  ☞  0.1948187
     (RANDOM)  ☞  0.7324636
```

(RANDOM) ☞ 0.6087399





4.10 Exemple D'utilisation Des Fonctions Numériques

```
? (DE QUAM (MSG) (PRIN1 MSG) (TERPRI -1) (READ))
= QUAM

? (DE FNT (X Y) (* NF (+ (COS (// X ND)) (+ (COS (// Y ND)) 2))))
= FNT

? (DE FRISES (NL NF ND X Y)
?     (PRINT "Quand tu en as marre fais ESCAPE.I .")
?     (WITH (ESCAPE.I NIL (&FINI 'VOILA))
?        (ESCLOOP &FINI
?           (SETQ
?              NL (QUAM "Donne le nb de lignes a faire (e.g. 60)")
?              NF (QUAM "Donne le facteur de * (e.g. 2.5)")
?              ND (QUAM "Donne le facteur de // (e.g. 7) "))
?           (SETQ Y 30.)
?           (REPEAT NL
?              (PRINC '/  4)
?              (SETQ X -30. Y (1- Y))
?              (REPEAT 61
?                 (PRINC
?                  ((FIX (FNT X Y))
?                   '(| - # + * /. : ~ ↑ , @ ~ ? /' + < > /[ /] { } )))
?                 (SETQ X (1+ X)))
?              (TERPRI)))))
= FRISES

?  (FRISES)
   "Quand tu en as marre fais ESCAPE.I ."
   "Donne le nb de lignes à faire (e.g. 60)" ?  25
   "Donne le facteur de * (e.g. 2.5)" ?  3.5
   "Donne le facteur de // (e.g. 7) " ?  7

    +##-----------##++**..:::~~~~↑↑↑~~~~:::..**++##-----------##+
    ##----|||||----##++**..:::~~~~~~~~~:::..**++##----|||||----##
    #---|||||||||---##++**..:::~~~~~~~:::..**++##---|||||||||---#
    #--|||||||||||--##++**...::::~~~::::...**++##--|||||||||||--#
    ---|||||||||||---##++**...:::::::::...**++##---|||||||||||---
    --|||||||||||||--##++**...:::::::::...**++##--|||||||||||||--
    --|||||||||||||--##++***...:::::::...***++##--|||||||||||||--
    --|||||||||||||--###++**...:::::::...**++###--|||||||||||||--
    --|||||||||||||--##++***...:::::::...***++##--|||||||||||||--
    --|||||||||||||--##++**...:::::::::...**++##--|||||||||||||--
    ---|||||||||||---##++**...:::::::::...**++##---|||||||||||---
    #--|||||||||||--##++**...::::~~~::::...**++##--|||||||||||--#
    #---|||||||||---##++**..:::~~~~~~~:::..**++##---|||||||||---#
    ##----|||||----##++**..:::~~~~~~~~~:::..**++##----|||||----##
    +##-----------##++**..:::~~~~↑↑↑~~~~:::..**++##-----------##+
    ++###-------###++**..:::~~~↑↑↑↑↑↑↑~~~:::..**++###-------###++
    *++###########++**..:::~~↑↑↑↑↑,↑↑↑↑↑~~:::..**++###########++*
    **+++#######+++**...::~~↑↑↑,,,,,,,↑↑↑~~::...**+++#######+++**
    .**+++++#+++++**...::~~↑↑,,,,,,,,,,,↑↑~~::...**+++++#+++++**.
    ..***+++++++***..:::~~↑↑,,,@@@@@@@,,,↑↑~~:::..***+++++++***..
    :..*****+*****..:::~~↑↑,,@@@@@@@@@@@,,↑↑~~:::..*****+*****..:
    ::...*******...:::~~↑↑,,@@@~~~~~~~@@@,,↑↑~~:::...*******...::
    :::....***....:::~~↑↑,,@@~~~~~~~~~~~@@,,↑↑~~:::....***....:::
    ~:::........:::~~↑↑,,@@~~~~?????~~~~@@,,↑↑~~:::........:::~
    ~~:::......:::~~↑↑,,@@~~~?????????~~~@@,,↑↑~~:::......:::~~
    ~~~::::...::::~~~↑↑,,@@~~???????????~~@@,,↑↑~~~::::...::::~~~
   "Donne le nb de lignes à faire (e.g. 60)" ?  r
```

CHAPITRE 5

LES CHAINES DE CARACTERES

VLISP permet la manipulation d'objets de type chaîne de caractères (STRINGS). Ces chaînes sont stockées dans une zone gérée dynamiquement. Si cette zone se révèle être de taille insuffisante, une erreur apparait ; le libellé de cette erreur est :

** no room for strings.

Cette erreur est fatale. Il faut augmenter la taille de la zone allouée aux chaînes, dans la fonction CONFIGURATION du fichier initial CONFIG.INI et relancer tout le travail.

Une chaîne de caractères est une suite illimitée (si ce n'est par la taille de la mémoire), de caractères quelconques. Sa représentation externe est cette suite de caractères encadrée du caractère délimiteur de chaîne (le caractère guillemet "). Si ce caractère doit être inséré dans une chaîne il doit être doublé.

Les caractéristiques d'une chaîne sont :
- les caractères qu'elle contient
- le nombre de ces caractères (la longueur de la chaîne).

ex :

la chaîne	contient les caractères	et a pour longeur
"CHer←10"	CHer←10	7
"(""NON""?)"	("NON"?)	8
""""	"	1
""	aucun	0
	(cette chaîne s'appelle la chaîne vide)	

En VLISP une chaîne de caractères est considérée comme un ATOME (et non une liste).

La valeur d'une chaîne de caractères est cette chaîne de caractères elle-même; il n'est donc pas nécéssaire de quoter les constantes de chaîne.

Tout comme les nombres, une chaîne n'a ni C-valeur ni P-liste.

```
ex : "galamantdelareine"    ☞  "galamantdelareine"
     (CAR "allatour")       ☞  NIL
     (ATOM "magnanime")     ☞  T
```

En entrée, la lecture des chaînes par la fonction READ est validée par le bit 17 du R.G. , ce bit est positionné par défaut.

En sortie, la restitution du caractère délimiteur de chaîne est validée par le bit 27 du R.G. , ce bit est positionné par défaut.

Le caractère délimiteur de chaîne (le caractère guillemet ") peut lui-même être modifié au moyen de la fonction (STATUS 16 <c>) dans laquelle <c> est le nouveau délimiteur de chaîne (cf: les fonctions STATUS).

Pour toutes les fonctions qui vont suivre, les arguments qui doivent être de type chaîne sont représentés par <strN>. Si ces arguments ne sont pas de type

chaîne, ils sont convertis automatiquement en chaîne au moyen de la fonction STRING.

5.1 Les Fonctions De Conversion De Chaînes

(STRING <s>) [SUBR à 1 argument]

convertit la S-expression <s> (de type quelconque) en une chaîne de caractères.
- si <s> est une chaîne STRING ramène cette même chaîne.
- si <s> = NIL, STRING ramène la chaîne vide "" .
- si <s> est un atome littéral, STRING ramène la chaîne contenant tous les caractères du P-name de cet atome.
- si <s> est un nombre, STRING ramène la chaîne contenant tous les caractères de la représentation externe de ce nombre dans la base de conversion courante.
- si <s> est une liste, elle est supposée être une liste de caractères (i.e. d'atomes mono-caractères) et STRING ramène la chaîne composée de ces caractères.

```
ex : (STRING "OUKELABONPOLIN")    ☞  "OUKELABONPOLIN"
     (STRING)                     ☞  ""
     (STRING 'RUSE)               ☞  "RUSE"
     (STRING (- 22.222 58.1))     ☞  "-35.878"
     (STRING '(A Z /. Q))         ☞  "AZ.Q"
```

(MAKLIST <str>) [SUBR à 1 argument]

convertit la chaîne <str> en une liste de caractères (i.e. d'atomes mono-caractères). Si <str> est la chaîne vide, MAKLIST ramène NIL.

```
ex : (MAKLIST "URSUS")     ☞  (U R S U S)
     (MAKLIST 'LAMBDA)     ☞  (L A M B D A)
```

(IMPLODE <s>) [SUBR à 1 argument]

si <s> est une chaîne, IMPLODE utilise les caractères constituant de cette chaîne de la même manière que s'ils étaient lus par la fonction READ. IMPLODE suppose donc que la chaîne de caractères est la représentation externe d'un objet lisp quelconque qui sera converti.
Si la chaîne de caractère n'est pas une représentation correcte d'un objet lisp, une erreur apparait ; le libellé de cette erreur est :
   ** IMPLODE error : <n>
dans lequel <n> est le numéro du type de l'erreur. Ces numéros sont identiques aux numéros de l'erreur ** READ error <n>.

```
ex : (IMPLODE "-567.9")                   ☞  -567.9
     (IMPLODE "()")                       ☞  NIL
     (IMPLODE "GENsymmm")                 ☞  GENsymmm
     (IMPLODE "'[A B . C]")               ☞  (QUOTE (MCONS A B C))
     (IMPLODE (CONCAT "(" "A B" ")"))     ☞  (A B)
     (IMPLODE "(A B C]")
        ** IMPLODE error : 6
```

(READSTR <n>) [SUBR à 1 argument] {pour READ STRing}

si l'argument <n> n'est pas fourni, ramène sous forme de chaîne, la ligne suivante du fichier d'entrée ; la ligne se termine à l'occurence du caractère RETURN qui n'est pas inclus dans la chaîne résultat. Cette fonction est souvent utilisée pour lire des réponses en langage naturel sur le terminal. Si l'argument numérique <n> est fourni, READSTR ramène les <n> caractères suivant du fichier d'entrée sous forme de chaîne. Cette autre possibilité est utilisée pour lire des fichiers de données en format fixe bloqué.

5.2 Prédicats Et Tests Sur Les Chaînes

(STRINGP <s>) [SUBR à 1 argument] {pour STRING Predicate}

ramène <s> si <s> est une chaîne, sinon STRINGP ramène NIL.

```
ex : (STRINGP "POT")      ☞   "POT"
     (STRINGP 56)         ☞   NIL
```

(NULLSTRP <str>) [SUBR à 1 argument] {pour NULL STRing Predicate}

ramène <str> si <str> est la chaîne vide "", sinon NULLSTRP ramène NIL. Il est à noter que NIL et la chaîne vide sont deux objets différents.

```
ex : (NULLSTRP (ADD1 5))  ☞  NIL
     (NULLSTRP NIL)       ☞  ""
       (car NULLSTRP convertit son argument)
```

(EQSTRING <str1> <str2>) [SUBR à 2 arguments]

ramène <str1> si la chaîne <str1> contient les mêmes caractères et possède la même longeur que la chaîne <str2>. Cette fonction est équivalente à la fonction EQ si les 2 arguments sont des chaînes; dans le cas où ils ne le seraient pas, EQSTRING va les convertir en chaîne avant d'effectuer la comparaison.

```
ex : (EQSTRING "ALA" "ALAS")          ☞  NIL
     (EQSTRING 'GLUCK "GLUCK")        ☞  "GLUCK"
     (EQSTRING '(A Z E R) 'AZER)      ☞  "AZER"
```

(STRINGL <str>) [SUBR à 1 argument] {pour STRING Length}

ramène le nombre de caractères de la chaîne <str>.

```
ex : (STRINGL "HUGH")       ☞  4
     (STRINGL (SUB1 -9))    ☞  3
     (STRINGL EXPR)         ☞  4
     (STRINGL NIL)          ☞  0
```

5.3 Fonctions De Création De Chaînes

(CONCAT <str1> ... <strN>) [SUBR à N arguments]

ramène une chaîne résultant de la concaténation des copies de toutes les chaînes <str1> ... <strN>.

ex : (CONCAT "Ab" (ADD1 14) "" 'AT) ☞ "Ab15AT"

(SUBSTRING <str> <n1> <n2>) [SUBR à 3 arguments]

ramène une copie de la sous-chaîne de <str> commençant à la position <n1> et se terminant à la position <n2>.
si <n1> > <n2>, SUBSTRING ramène la chaîne vide "".
si <n1> n'est pas un nombre, la sous-chaîne commence au debut (i.e. en position 1) de la chaîne <str>.
si <n2> n'est pas un nombre ou est omis, la sous-chaîne se termine à la fin de la chaîne <str>.

```
ex : (SUBSTRING "ABRACAD" 2 4)    ☞ "BRA"
     (SUBSTRING "ABRACAD" () 4)   ☞ "ABRA"
     (SUBSTRING "ABRACAD" 2)      ☞ "BRACAD"
     (SUBSTRING 'LONGMOT 4 4)     ☞ "G"
```

(DUPL <str> <n>) [SUBR à 2 arguments]

ramène la chaîne résultant de la duplication <n> fois de la chaîne <str>. Si <n> n'est pas un nombre ou est omis, DUPL ramène une copie de la chaîne <str>. Si <n> est <= à 0, DUPL ramène la chaîne vide "".

```
ex : (DUPL "ALO" 3)    ☞ "ALOALOALO"
     (DUPL 'ALO)       ☞ "ALO"
     (DUPL "ALO" -1)   ☞ ""
```

(REVERSTR <str>) [SUBR à 1 argument]

ramène une copie de la chaîne <str> inversée.

```
ex : (REVERSTR 'POTOP)      ☞ "POTOP"
     (REVERSTR "DELUGE")    ☞ "EGULED"
```

(TRANSLATE <str1> <str2> <str3>) [SUBR à 3 arguments]

ramène une copie de la chaîne <str1> en y remplaçant les caractères qui font partie de la chaîne <str2> par leurs homologues dans la chaîne <str3>. Si des caractères de la chaîne <str2> n'ont pas d'homologues dans la chaîne <str3> (i.e. si (STRINGL <str2>) > (STRINGL <str3>) alors ces caractères sont enlevés de la chaîne résultante.

ex : (TRANSLATE "ABRACADABRA" "ARC" "OL") ☞ "OBLOODOBLO"
se lit : traduire la chaîne ABRACADABRA en y remplacant tous les A par O, tous les R par L et en y enlevant tous les C.

(NEXTSTR <str>) [SUBR à 1 argument] {pour NEXT STRing}

cette fonction est la seule fonction qui modifie physiquement la chaîne argument <str>. NEXTSTR ramène en valeur le premier caractère de la chaîne <str> et l'enlève PHYSIQUEMENT de celle-ci. NEXTSTR est donc l'équivalent pour les chaînes de la fonction NEXTL qui opère sur des listes. Cette fonction est utilisée pour obtenir un à un les différents caractères d'une chaîne.

```
ex : (SETQ strg "ESARINTULOC")  ☞  "ESARINTULOC"
     (NEXTSTR strg)             ☞  "E"
     strg                       ☞  "SARINTULOC"
     (NEXTSTR strg)             ☞  "S"
     strg                       ☞  "ARINTULOC"
```

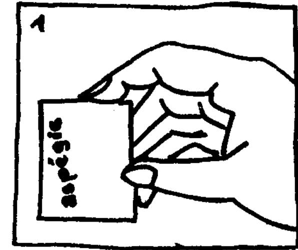
5.4 Exemples D'utilisation Des Chaînes

```
?
?  (DE CONVBDC (NB)
?       (IF (LE NB 1)
?             NB
?             (CONCAT (CONVBDC (QUO NB 2))
?                       (REM NB 2))))
=  CONVBDC
=  ; time = 0 ms ;
?
?  (CONVBDC 13)
=  "1101"
=  ; time = 0 ms ;
?
?
?  (DE PALINDROM (CH)
?       ; fonction qui teste si la chaîne CH est un palindrome ;
?       (SETQ CH (TRANSLATE CH ",;.:!?'()[] "))
?       ; enlève tous les espaces et les signes de ponctuation ;
?       (EQSTRING (REVERSTR CH) CH))
=  PALINDROM
=  ; time = 0 ms ;
?
?  (PALINDROM "ELU PAR CETTE CRAPULE ...")
=  "ELUPARCETTECRAPULE"
=  ; time = 20 ms ;
?
?  (PALINDROM "GALAMANTELAREINE")
=  NIL
=  ; time = 20 ms ;
?
?  ←
Bye
.
```



CHAPITRE 6

LES TABLEAUX ET LA PILE UTILISATEUR



Un nouveau type d'objet est apparu : le tableau . Un tableau est un ensemble de mémoires contigues pouvant contenir n'importe quel objet VLISP (atome, nombre, chaîne ou liste). Les différents éléments d'un tableau sont numérotés à l'aide d'un indice. Tous les tableaux sont actuellement uni-dimensionnés, le premier indice est 0 (ces tableaux sont donc des vecteurs).

Les différents tableaux sont stockés dans une zone spéciale allouée statiquement ; cette zone est partagée avec la pile utilisateur. Les fonctions sur les tableaux permettent de définir un tableau, d'avoir accès aux éléments d'un tableau, de modifier un élément d'un tableau ou d'appliquer une fonction aux différents éléments d'un tableau.

6.1 Définition Et Accès Aux Tableaux

(DA <nom> <taille> <fonction>) [SUBR à 3 arguments]
{pour Define Array}

permet de définir un nouveau tableau. <nom> est un atome littéral qui devient le nom du tableau; <taille> est le nombre d'éléments que peut contenir le tableau. S'il n'y a pas assez de place pour stocker ce nouveau tableau, l'erreur suivante apparait :

** no room for arrays.

Si la fonction (d'initialisation) n'est pas fournie (i.e. si <fonction>=NIL), tous les éléments du tableau sont initialisés à NIL; dans le cas contraire, la fonction est appliquée pour chaque élément avec l'indice de cet élément lié au 1er argument (s'il existe) de la fonction. Il peut donc y avoir un appel implicite de la fonction MAPARRAY. Les différentes valeurs ramenées par cette fonction seront les valeurs d'initialisation des éléments du tableau. DA ramène le nom du tableau en valeur.

```
ex : (DA 'TB 100)          ; définition d'un tableau de nom TB de
                              100 éléments. Chaque élément est
                              initialisé à NIL ;

     (DA 'TBL 10  (LAMBDA (X) 0)) ; définition d'un tableau de
                                    10 éléments, chaque élément
                                    est initialisé à 0 ;

     (DA 'ARR 10             ; définition d'un tableau de 10
        (LAMBDA (X) (- 9 X)))   éléments initialisés à :
                                9 8 7 6 5 4 3 2 1 0 ;
```

Une fois le tableau défini, on a accès à chacun des éléments du tableau de deux manières différentes :

- en évaluant une forme à 1 argument dont la fonction est le nom du tableau et l'argument l'indice de l'élément désiré. Le nom du tableau devient donc une fonction d'accès à ce tableau. L'argument de cette fonction d'accès est évalué.

- en utilisant la fonction ARRAY.

(ARRAY <at> <n>) [SUBR à 2 arguments]

ramène la valeur de l'élément <n> du tableau <at>.
(ARRAY '<at> <n>) est donc équivalent à (<at> <n>) mais oblige à ce que <at> soit un nom de tableau précédemment défini.

L'interprète ne teste la validité de l'indice fourni que si le bit 7 du R.G. est positionné (ce qui est l'option par défaut).

Il est facile de définir de nouvelles fonctions d'accès à des tableaux déjà définis ce qui permet de tester soi-même la validité du ou des indices (voir les derniers exemples).

```
ex : (TB 5)                  ☞ NIL
     (TBL (ADD1 4))          ☞ 0
     (ARRAY 'TBL (ADD1 4))   ☞ 0
```

6.2 Les Fonctions Standard Sur Les Tableaux

Pour toutes les fonctions sur les tableaux qui vont être décrites, les erreurs (le nom fourni n'est pas un nom de tableau ou l'indice est incorrect) sont signalées par le message :
** array error : <nom du tableau> <indice>

(DIM <nom>) [SUBR à 1 argument]

l'argument <nom> doit être un nom de tableau. DIM ramène le plus grand indice de ce tableau.

```
ex : (DA 'TB 50)   ☞ TB
     (DIM 'TB)     ☞ 49
```

(SETA <nom> <indice> <valeur>) [SUBR à 3 arguments]
{pour SET Array}

met dans l'élément du tableau spécifié par le nom et l'indice, la valeur fournie en troisième argument. SETA ramène en valeur la valeur chargée.

```
ex : (SETA 'TB 5 10)  ☞ 10
     (TB 5)           ☞ 10
```

(SETQA <nom> <indice> <valeur>) [FSUBR]

est identique à la fonction SETA mais le nom du tableau n'est pas évalué, en revanche l'indice <indice> et la valeur <valeur> le sont toujours.

```
ex : (SETQA TB 6 (ADD1 19))  ☞ 20
```

```
(TB 6)                       ☞ 20
```

(MAPARRAY <nom> <fonction>) [SUBR à 2 arguments]

permet d'appliquer la fonction spécifiée à chacun des éléments du tableau dont le nom est donné en 1er argument. Le 1er argument de la fonction est lié (s'il existe) à l'indice de l'élément du tableau. MAPARRAY ramène NIL en valeur.

```
ex:  ; initialisation des différents éléments de TB avec
       les valeurs 1 2 3 4 5 .. ;
     (MAPARRAY 'TB (LAMBDA (X) (SETA 'TB X (ADD1 X))))
      ☞ NIL
```

(MAPARRAYQ <nom> <fonction>) [FSUBR]

est identique à la fonction MAPARRAY mais le nom du tableau n'est pas évalué, en revanche le nom de la fonction l'est toujours.

(FILLARRAY <nom> <liste>) [SUBR à 2 arguments]

range dans les différents éléments du tableau, dont le nom est spécifié comme 1er argument, les éléments successifs de la liste 2ème argument. Si la liste est trop courte, le reste du tableau est rempli avec des NIL ; si la liste 2ème argument est en réalité un atome, le tableau est rempli avec cet atome. FILLARRAY ramène le nom du tableau en valeur.

```
ex : ; définition d'un tableau ;
     (DA 'TB 10)                       ☞ TB
     ; initialisation des éléments aux valeurs spécifiées ;
     (FILLARRAY 'TB '(0 2 4 6 5 3 1))  ☞ TB
     ; initialisation de tous les éléments à la valeur 99 ;
     (FILLARAY 'TB 99)                 ☞ TB
```

(LISTARRAY <nom>) [SUBR à 1 argument]

ramène la liste constituée de tous les éléments du tableau dont le nom est fourni en 1er argument.

```
ex : (DA 'TB 5)                  ☞ TB
     (FILLARRAY 'TB '(10 11 12)) ☞ TB
     (LISTARRAY 'TB)             ☞ (10 11 12 NIL NIL)
```

6.3 Exemples D'utilisation Des Tableaux

```
? (DA 'TB 100)          ; définition d'un tableau de nom TB   ;
= TB                    ; possédant 100 éléments, chacun des  ;
                        ; éléments est initialisé à NIL.      ;
? (DIM 'TB)
= 99

? (DE TB1 (I)           ; définition d'une nouvelle fonction  ;
?   (IF (LE 0 I 99)     ; d'accès à TB1 qui teste la validité ;
?       (TB I)          ; de l'indice fourni ;
?       (ERROR ["débordement TB1 " I])))
= TB1
```

```
? (DE TB2 (I J)          ; définition d'une nouvelle fonction ;
?   (IF (AND (LE 0 I 9)    ; permettant d'accèder au tableau  ;
?            (LE 0 J 9))   ; TB au moyen de 2 indices pouvant ;
?       (TB (PLUS (TIMES I 10) J))  ; varier de 0 à 9. Cette ;
?       (ERROR ["débordement TB2 " I J])))  ; fonction teste ;
                                         ; les indices fournis ;
= TB2

? (SETQA TB 20 3)
= 3

? (TB1 20)
= 3

? (TB2 2 0)
= 3

? (DA 'CLAIR 10 (LAMBDA (X)
?      ((ADD1 X) '("zero" "un" "deux" "trois" "quatre"
?                  "cinq" "six" "sept" "huit" "neuf"))))
= CLAIR

? (CLAIR 1)
= "un"

? (CLAIR 2)
= "deux"

? (MAPARRAYQ CLAIR (LAMBDA (X)
?     (SETQA CLAIR X (CONCAT "-" (CLAIR X) "-"))))
= NIL

? (CLAIR 9)
= "-neuf-"
```

6.4 La Pile Utilisateur

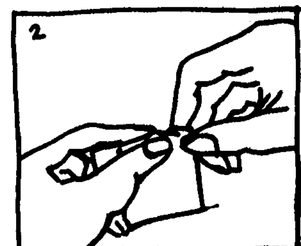
Il existe une zone mémoire dans laquelle il est possible de sauver temporairement des objets VLISP quelconques. Cette zone est gérée comme une pile : c'est la pile utilisateur (qui n'a aucun rapport avec la pile utilisée par l'interprète). La zone dans laquelle est stockée cette pile utilisateur étant partagée avec les tableaux, sa taille est donc déterminée par la différence entre la taille de la zone allouée aux tableaux et la somme des tailles de tous les tableaux définis. En cas de débordement de pile les erreurs suivantes apparaissent :

** user stack overflow.
** user stack underflow.

Après ces erreurs (comme après toute autre erreur) le pointeur de pile utilisateur est reinitialisé.

(PUSH <s1> ... <sN>) [SUBR à N arguments]

empile les valeurs des differentes expressions <s1> ... <sN> et raméne <sN> en valeur. Toutefois si aucun argument n'est fourni (i.e. si on évalue (PUSH)), une valeur NIL est quand même empilée.



(POP <n>) [SUBR à 1 argument]

si l'argument <n> n'est pas fourni, le sommet de pile est ramené en valeur et le pointeur de pile est mis à jour. Si l'argument numérique <n> est donné, POP ramène le <n>ième élément contenu dans la pile à partir du pointeur de pile mais celui-ci n'est pas modifié. Si ce nombre est négatif, on peut accèder à des éléments de la pile précédemment empilés puis dépilés. Il ne faut toutefois pas sortir de la zone allouée pour la pile sous peine de déclencher les erreurs de débordement de pile.

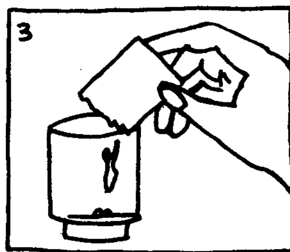
(PSTACK <n>) [SUBR à 1 argument]

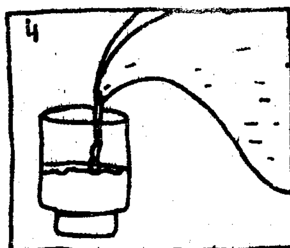
si l'argument <n> n'est pas fourni, ramène le pointeur de pile courant. Si l'argument numérique <n> est fourni, il devient la nouvelle valeur du pointeur de pile. Cette fonction sert donc tout naturellement à manipuler le pointeur de pile lui-même.

Exemples d'utilisation de la pile utilisateur

```
ex : (PUSH 'A 3)          ☞ 3
     (PUSH)               ☞ NIL
     (PUSH 'B)            ☞ B
     (POP 0)              ☞ B
     (POP 1)              ☞ NIL
     (POP 2)              ☞ 3
     (POP)                ☞ B
     (POP)                ☞ NIL
     (POP 1)              ☞ A
     (POP -2)             ☞ B
     (POP)                ☞ 3
     (POP)                ☞ A

     (PUSH 1 2)               ☞ 2
     (SETQ PPILE (PSTACK))    ☞ 33456    (par exemple)
     (PUSH 3 4 5 6 7 8)       ☞ 8
     (PSTACK PPILE)           ☞ 33456
     (POP)                    ☞ 2
     (POP)                    ☞ 1
     (POP)
   ** user stack underflow.
      ... etc ...
```




4

CHAPITRE 7

LES ENTREES / SORTIES ET LES FICHIERS

7.1 Les Fonctions D'entrée De Base

Toutes ces fonctions utilisent le fichier d'entrée sélecté par la fonction INPUT ou FILOP.
Tous les fichiers d'entrée sont compatibles avec l'éditeur de l'IRCAM ETV. La première page du fichier, qui contient son répertoire et qui est mis à jour par l'éditeur, est considérée par la fonction d'entrée READ comme un commentaire. Il faut toutefois prendre garde à ne pas mettre un caractère C en tête d'un fichier; ce caractère C étant justement, pour VLISP, la marque du répertoire de ETV.
Si le fichier d'entrée est associé à un terminal, l'éditeur de ligne standard (DEC ou IRCAM) est activé, le caractère ? est imprimé avant toute lecture de ligne et un petit Pretty-print effectue un renfoncement des entrées en fonction de la profondeur de la lecture.

(READ) [SUBR à 0 argument]

lit la S-expression suivante (atome ou liste) du fichier d'entrée et la ramène en valeur. READ est la principale fonction de lecture.
Si une erreur de syntaxe est détectée, une erreur fatale se produit et le libellé suivant est imprimé :

    ** read error <n>

dans lequel <n> est le code de l'erreur. Ce code peut être :

1 la S-expression débute par un point (.)
2 la S-expression débute par)
3 la S-expression débute par]
4 un . ne se trouve pas avant le dernier élément d'une liste
5 mélange de séparateurs du genre (x y . z]
6 occurence d'un] sans [correspondant
7 occurence d'une) sans (correspondante
8 fin de liste de IMPLODE
9 argument incorrect pour IMPLODE

(READCH) [SUBR à 0 argument] {pour READ CHaracter}

lit et ramène en valeur le caractère suivant du fichier d'entrée. Ce caractère est retourné sous la forme d'un atome (littéral pour les lettres et numérique pour les chiffres).

(PEEKCH) [SUBR à 0 argument] {pour PEEK CHaracter}

ramène en valeur le caractère du fichier d'entrée d'une manière identique à la fonction READCH, toutefois, ce caractère n'est pas véritablement lu mais seulement "consulté". Il en résulte que des appels successifs de la fonctions PEEKCH ramènent toujours le même résultat. PEEKCH est utilisé pour "sentir" le caractère suivant du fichier d'entrée AVANT de le lire véritablement.

(TEREAD) [SUBR à 0 argument] {pour TErminate READ}

passe à l'enregistrement suivant du fichier d'entrée (une ligne sur TTY, une carte perforée, ...) en ignorant le reste éventuel de l'enregistrement courant. TEREAD ramène NIL en valeur.

(READSTR <n>) [SUBR à 1 argument] {pour READ STRing}

si l'argument <n> n'est pas fourni, ramène l'enregistrement suivant du fichier d'entrée sous forme d'une chaîne de caractères, un enregistrement étant une suite de caractères quelconques encadrée d'un (ou plusieurs) caractères de type NULL. Ce type d'utilisation est utile par exemple pour lire sur le terminal des lignes sous forme de chaînes de caractères. Si l'argument numérique <n> est donné, READSTR ramène la chaîne constituée des <n> caractères suivants du fichier. Tous les caractères de type NULL sont sautés dans ce type de lecture.
Cette fonction est utilisée pour lire des fichiers de données en format fixe ou variable.

7.2 Contrôle Des Fonctions D'entrée

7.2.1 La Lecture Standard -

La lecture des S-expressions VLISP s'effectuent en format LIBRE.

Durant la lecture des atomes litéraux seuls les 13 premiers caractères sont pris en compte. Il y a transcodage automatique des caractères minuscules en caractères majuscules (ce transcodage est contrôlé par le bit 19 du R.G.). S'il faut insérer des caractères spéciaux dans un atome littéral, il faut les faire précéder du caractère quote-caractère /.

ex :	CaR	correspond à l'atome		CAR
	LONGTRESLONGATOME	"	"	LONGTRESLONGA
	RE/(3/)	"	"	RE(3)

Les nombres sont représentés par une suite de digits dans la base d'entrée courante et peuvent être précédés du signe - .

Les chaînes de caractères sont représentées par un suite de caractères quelconques encadrée du caractère délimiteur de chaîne (par défaut le "). N'importe quel caractère peut faire partie d'une chaêne en particulier les caractères Return et Line-feed. Il n'y a jamais de transcodage minuscule majuscule durant la lecture des chaînes.

Il est possible d'insérer des commentaires, qui sont totalement ignorés au cours de la lecture. Un commentaire est une suite de caractères quelconques encadrée du caractère délimiteur de commentaire (par défaut le ;).

La représentation des listes est classique : une liste se représente par une parenthèse ouvrante "(" suivie des éléments de la liste suivis d'une

parenthèse fermante ")". Il est possible également d'utiliser la notation pointée généralisée.

```
ex :   (A . (B . (C . D))) correspond à (A B . C)
       ((A) . (B))              "     "   ((A) B)
```

Il existe en VLISP une nouvelle notation permettant de décrire les appels des fonctions de constructions de listes simples NCONS CONS MCONS et LIST. Cette notation utilise les crochets carrés [ ] et le point.

```
[ s ]                    correspond à    (NCONS s)
[ s1 . s2 ]                    " "       (CONS s1 s2)
[ s1 s2 ,,, sN-1 . sN ]        " "       (MCONS s1 s2 ,,, sN-1 sN)
[ s1 s2 ,,, sN ]               " "       (LIST s1 s2 ,,, sN)
```

```
ex :  ['PREM  Y [(CDDR L) . [X]]]  est équivalent à
      (LIST 'PREM Y (CONS (CDDR L) (NCONS X)))
```

Au début de la fonction READ, il peut y avoir un nombre quelconque de parenthèses fermantes qui sont ignorées. Ceci permet de refermer à coup sûr les S-expressions avec une giclée de parenthèses fermantes sans avoir à les dénombrer.

```
(DE FOO (N) (ADD1 N)))))) est lu sans erreur
```

Des bits du R.G. permettent de régler certaines modalités de lecture en particulier :

- le bit 10 du R.G. fait imprimer l'image de tous les enregistrements lus en entrée. Ce bit n'est pas positionné par défaut.

- le bit 12 du R.G. valide le signe + au début des nombres. Ce bit n'est pas positionné par défaut.

- le bit 13 du R.G. valide le signe - au début des nombres. Ce bit est positonné par défaut.

- le bit 14 du R.G. valide l'action du QUOTE-caractère (par défaut le /). Ce bit est positionné par défaut.

- le bit 17 du R.G. valide la lecture des chaînes de caractères. Ce bit est positionné par défaut.

- le bit 18 du R.G. valide la lecture des commentaires encadrés du delimiteur ; . Ce bit est positionné par défaut.

- le bit 19 du R.G. valide le transcodage minuscule majuscule en entrée. Ce bit est positionné par défaut.

7.2.2 Les Macro-caractères -

Un macro-caractère est un caractère auquel est associé une fonction qui est lancée automatiquement à la lecture de ce caractère dans le flux d'entrée. La valeur ramenée par cette fonction est insérée à la place du macro-caractère. Tous les caractères peuvent être utilisés comme macro-caractère. L'interprétation des macro-caractères dans le flux d'entrée est validée par le bit 15 du R.G.

Il existe 2 macro-caractères standard :
- le quote (apostrophe) ' qui, placé devant une S-expression quelconque,

ramène la liste (QUOTE expression).
ex : '(A B) est équivalent à (QUOTE (A B))
''A " (QUOTE (QUOTE A))

- l'anti-slash \ qui, placé, devant une S-expression quelconque ramène cette S-expression en considérant que tous les nombres qui y sont inclus sont écris en octal.
ex : \(630 . 300) est équivalent à (408 . 192) en décimal

(DMC <c> <l> <s1> ... <sN>) [FSUBR]
{pour Define Macro Character}

l'argument <c> doit être un atome mono-caractère. DMC associe à ce caractère une fonction qui possède une liste de variables locales <l> (cette liste est obligatoire à cette position même s'il n'y a pas de variables locales) et un corps de fonction <s1> ... <sN>. DMC ramène <c> en valeur.

ex : si les macro-caractères standard n'étaient pas définis, on pourrait le faire de la manière suivante :

```
(DMC /' ()
   [QUOTE (READ)])        ; ramène la liste (QUOTE <s-expr lue>) ;

(DMC /\ ()
   (PUSH (STATUS 5))      ; sauve l'ancienne base d'entrée ;
   (STATUS 5 8)           ; passage en mode octal ;
   (PROG1
      (READ)              ; lecture de l'expression
                          (qui est ramenée en valeur) ;
      (STATUS 5 (POP))))  ; restauration de l'ancienne base ;
```

(STATUS 18 <c> <s>) [SUBR à N arguments]

permet également de définir un macro-caractère. l'argument <c> (évalué) doit être un caractère, et l'argument <s> doit être une fonction explicite (LAMBDA-expression). Ce STATUS permet de définir des macro-caractères dont le caractère est évalué. Si <s> est donné cette fonction ramène <s> sinon cette fonction ramène la définition courante associée au caractère <c>.

ex :
```
(STATUS 18 (ADD1 \136)  ; c'est le caractère '←' ;
      (LAMBDA () ['STOP])) ; qui provoque l'arret de VLISP ;
```

Si la fonction DMC n'était pas standard on pourrait la définir au moyen de la fonction STATUS :

```
(DF DMC (L)
   (STATUS 18
      (CAR L)
      ['LAMBDA . (CDR L)])
   (CAR L))
```

(STATUS 19 <c>) [SUBR à N arguments]

détruit la définition du macro-caractère <c> (plus précisément enlève la fonction associée à ce macro-caractère). Si <c> n'est pas un caractère, provoque l'erreur STATUS. Cette fonction ramène <c> en valeur.

(STATUS 20) [SUBR à N arguments]

ramène un pointeur sur la dernière S-expression lue avant l'appel d'un macro-caractère. Ce STATUS est utilisé pour connaître (CAR (STATUS 20)) ou pour modifier (SET (STATUS 20) ...) la dernière S-expression lue juste avant l'appel d'un macro-caractère.

ex : création des 2 macro-caractères
":" pour CONS et ":=" pour SETQ en notation infixe

```
(DMC ":" ()
   (PUSH (CAR (STATUS 20)))              ; pour le ramener ;
   (SET (STATUS 20)                      ; modifie l'élément précédent ;
      (IF (NEQ (PEEKCH) '/=) 'CONS       ; avec CONS si : ;
              (READCH) 'SETQ))           ; ou SETQ si := ;
   (POP))                                ; ramène l'élément précédent ;
```

```
la lecture de (A : B) sera équivalente à (CONS A B)
     "        (A B : C)        "         (A CONS B C)
     "        (X := 4)         "         (SETQ X 4)
```

7.2.3 Type Des Caractères Et READ.TABLE -

L'analyseur lexical VLISP (i.e. la fonction READ) utilise une "table de lecture" pour effectuer commodemment son analyse. Cette table associe un type code ainsi qu'une valeur en tant que macro-caractère à chacun des caractères. Cette table de lecture est totalement accessible à l'utilisateur qui peut ainsi la changer pour pouvoir lire facilement de nouveaux dialectes de LISP aux syntaxes étranges.

Les types disponibles sont les suivants :

0 : type NULL. Tous les caractères de ce type sont complètement ignorés à la lecture (Ex: le caractère Line/Feed, le caractère "Avance-bande" ou NULL ...).

1 : type SEP. Définit un caractère séparateur standard (Ex: l'espace, la tabulation ...).

2 : type NORMAL. Définit un caractère normal pouvant être utilisé pour construire un P-name.

3 : type DOT. Ce type de caractère (le point . par défaut) sert à écrire les paires pointées.

4 : type LPAR. Ce type de caractère (la parenthèse ouvrante (par défaut) sert de caractère de début de liste.

5 : type RPAR. Ce type de caractère (la parenthèse fermante) par défaut) sert de caractère de fin de liste.

6 : type LBRA. Ce type de caractère (le crochet ouvrant [par défaut) sert de caractère de début d'appel de la fonction LIST.

7 : type RBRA. Ce type de caractère (le crochet fermant] par défaut) sert de caractère de fin d'appel de la fonction LIST.

8 : type COMMENT. Ce type de caractère sert à indiquer le début d'un commentaire qui sera terminé à l'occurence d'un caractère de ce même type. Par défaut il n'existe qu'un caractère de ce type, le caractère point virgule ; .

9 : type CSTRING. Ce type de caractère fait office de caractère délimiteur de chaîne. Par défaut il n'existe qu'un seul caractère de ce type, le caractère guillemets " .

10 : type QUOTEC. Ce type de caractère est utilisé pour "quoter" n'importe quel autre caractère. "Quoter" un caractère consiste à lui donner implicitement le type 2 (le type des caractères normaux). Par défaut il n'existe qu'un seul caractère de ce type, le caractère "slash" / .

7.2.3.1 Manipulation Des Types -

Un certain nombre de fonctions standard et de fonctions de type STATUS vont pouvoir manipuler cette table. Les STATUS 14, 15, et 16 des versions précédentes de VLISP (VLISP 10 et VLISP 10.2) ne sont plus utilisés mais sont conservés dans le système pour des raisons de compatibilité.

(STATUS 17 <c> <n>) [SUBR à N arguments]

Ce STATUS permet de connaître et/ou de modifier le type du caractère <c>. Si <n> est donné, le type est modifié. Cette fonction ramène le type courant du caractère <c> après modification éventuelle.
Si l'argument <c> n'est pas un caractère (i.e. un atome de type quelconque dont le P-Name a pour longeur 1) ou si l'argument <n> est fourni mais n'est pas un nombre, une erreur apparait dont le libellé est :
** STATUS ERROR : TYPECH.

(TYPECH <c> <n>) [FSUBR] {pour TYPE CHaracter}

Cette fonction est identique à la fonction STATUS 17 mais les arguments ne sont pas évalués.

ex : (STATUS 17 '/.) ☞ 3

(TYPECH < 4) ☞ 4
(TYPECH > 5) ☞ 5
l'entrée <CONS '(A> '<B)> sera lue (CONS '(A) '(B))

7.2.3.2 Manipulation De La Table De Lecture -

Ces deux fonctions manipulent la table de lecture d'une manière globale. Une table de lecture en VLISP est une liste de 128 éléments (il y a donc un élément par caractère). Chaque élément peut être :
- un nombre qui représente le type du caractère
- une liste dont le CAR est un nombre qui représente le type du caractère et dont le CDR est la fonction associée à ce caractère considéré comme un macro-caractère. Cette fonction est soit une lambda-expression explicite soit un nombre dénotant l'adresse de la fonction standard associée à ce caractère.

(READ.TABLE <l>) [SUBR à 1 argument]

Si l'argument <l> est fourni, il représente la nouvelle table de lecture (sous le format décrit précédemment). READ.TABLE ramène la table de lecture courante aprés modification éventuelle.

(READ.STD) [SUBR à 0 argument] {pour READ STanDard}

remet la table de lecture dans son état initial (état de la table au départ de l'interprète). READ.STD ramène en valeur cette table de lecture originelle sous le format décrit précédemment.

ex : si un programme fait une large utilisation des fonctions de modification de la table de lecture, il est prudent de le faire précéder de quelquechose du genre :

```
(PUSH (READ.TABLE))
```

et de le faire se terminer par :

```
(READ.TABLE (POP))
```

Un certain nombre de programmes systèmes qui aiment se faire lire par tous et dans n'importe quelles circonstances débutent par :

```
(PROGN (SETQ READ.TABLE (READ.TABLE))
       (READ.STD)
       'PRET)
```

et se terminent par :

```
(READ.TABLE READ.TABLE)
```

Voici l'édition de la table normale de lecture :

```
(DE  /→ (N) ['/→ N])
(DMO /→ (N) (TTAB N))
(DMC /→ ()  ['/→ (READ)])

(DE PRTABL (N L)
   (WHILE L
      (PRINC '/\)
      (STATUS 6 8)
      (PRIN1 N)
      (STATUS 6 10)
      (PRIN1 →8 N →13 (ASCII N) →16)
      (INCR N)
      (IF (ATOM (CAR L)) (PRINT (CAR L))
        (PRINT (CAAR L) →20 (CDAR L)))
      (NEXTL L)))

(PRTABL 0 (READ.TABLE))
```

\0	0		0	
\1	1	â	2	(LAMBDA NIL ['LIBRARY (READ)])
\2	2	à	2	
\3	3	┌	2	
\4	4	ê	2	
\5	5	é	2	(LAMBDA NIL (RUN '(SYS (E . SHR)) -1))
\6	6	è	2	(LAMBDA NIL ['PHENARETE (READ)])
\7	7	┼	2	(LAMBDA NIL (DISPLAY '(127 7)) '┼)
\10	8	λ	1	(LAMBDA NIL 'LAMBDA)
\11	9		1	
\12	10			
0				
\13	11	∫	1	
\14	12			
1				
\15	13			
0				
\16	14	ô	2	
\17	15	¯	2	
\20	16	┌	2	(LAMBDA NIL ['PRETTY (READ)])
\21	17	┐	2	(LAMBDA NIL (STATUS 1 5) (STATUS 21) (STATUS 2 5) NIL)
\22	18	ù	2	
\23	19	î	2	
\24	20	û	2	
\25	21	⌉	2	
\26	22	⌊	2	
\27	23	┘	2	(LAMBDA NIL (RUN '(SYS (WHO . SAV))))
\30	24	_	2	
\31	25	→	2	(LAMBDA NIL ['/→ (READ)])
\32	26	≠	2	
\33	27	~	2	
\34	28	🅅	2	
\35	29	🄻	2	
\36	30	🅂	2	
\37	31	🄿	2	
\40	32		1	
\41	33	!	2	
\42	34	"	9	
\43	35	#	2	
\44	36	$	2	
\45	37	%	2	
\46	38	&	2	
\47	39	'	2	132836
\50	40	(	4	
\51	41	)	5	
\52	42	*	2	
\53	43	+	2	
\54	44	,	2	
\55	45	-	2	
\56	46	.	3	
\57	47	/	10	
\60	48	0	2	
\61	49	1	2	
\62	50	2	2	
\63	51	3	2	
\64	52	4	2	
\65	53	5	2	
\66	54	6	2	
\67	55	7	2	
\70	56	8	2	
\71	57	9	2	
\72	58	:	2	
\73	59	;	8	

\74	60	<	2	
\75	61	=	2	
\76	62	>	2	
\77	63	?	2	
\100	64	@	2	
\101	65	A	2	
\102	66	B	2	
\103	67	C	2	
\104	68	D	2	
\105	69	E	2	
\106	70	F	2	
\107	71	G	2	
\110	72	H	2	
\111	73	I	2	
\112	74	J	2	
\113	75	K	2	
\114	76	L	2	
\115	77	M	2	
\116	78	N	2	
\117	79	O	2	
\120	80	P	2	
\121	81	Q	2	
\122	82	R	2	
\123	83	S	2	
\124	84	T	2	
\125	85	U	2	
\126	86	V	2	
\127	87	W	2	
\130	88	X	2	
\131	89	Y	2	
\132	90	Z	2	
\133	91	[	6	
\134	92	\	2	132844
\135	93	]	7	
\136	94	↑	2	
\137	95	←	2	(LAMBDA NIL (STOP))
\140	96	ç	2	
\141	97	a	2	
\142	98	b	2	
\143	99	c	2	
\144	100	d	2	
\145	101	e	2	
\146	102	f	2	
\147	103	g	2	
\150	104	h	2	
\151	105	i	2	
\152	106	j	2	
\153	107	k	2	
\154	108	l	2	
\155	109	m	2	
\156	110	n	2	
\157	111	o	2	
\160	112	p	2	
\161	113	q	2	
\162	114	r	2	
\163	115	s	2	
\164	116	t	2	
\165	117	u	2	
\166	118	v	2	
\167	119	w	2	
\170	120	x	2	
\171	121	y	2	
\172	122	z	2	



\173	123	{	2
\174	124	|	2
\175	125	}	2
\176	126	~	2
\177	127		0

7.2.4 Les Macro-fonctions D'entrée : MACIN -

Si au cours d'une lecture la fonction READ lit une liste dont le CAR, atomique, possède sur sa P-liste l'indicateur MACIN, la fonction associée à cet indicateur est lancée automatiquement avec pour liste d'arguments le CDR de la liste qui a été lue. La valeur ramenée par cet appel remplace le liste qui a été lue. Ces fonctions sont appellées des Macro fonctions d'entrée et sont définies au moyen de la fonction DMI.
Il existe un indicateur, le bit 16 du R.G. qui permet de valider le traitement de ce type de fonctions. S'il est positionné (ce qui est l'option par défaut) ces fonctions sont traitées.

(DMI <at> <l> <s1> .. <sN>) [FSUBR]
{pour Define Macro Input}

définit une macro fonction d'entrée associée à l'atome littéral <at>, la liste des arguments de cette fonction est <l> et le corps de cette fonction est <s1> ... <sN>. DMI ramène <at> en valeur et pourrait être définie en VLISP :

```
(DF DMI (l)
   (PUT (CAR l) (CONS 'LAMBDA (CDR l)) 'MACIN))
```

7.3 Le Mode LIBRARY

Un nouveau mode de chargement rapide a été ajouté : le mode LIBRARY, qui, silencieusement, vous charge n'importe quel fichier. Ce fichier (d'extension VLI, VLA ou VLO) peut se trouver dans votre zone disque ou bien dans d'autres zones que vous pouvez spécifier grâce à la nouvelle fonction PATHLIBRARY.
ATTENTION : les fonctions se trouvant dans ce type de fichier ne doivent pas contenir d'erreurs de syntaxe (de lecture), ce mode est réservé aux chargements rapides des fichiers systèmes et des fichiers contenant les fonctions AUTOLOAD (cf: l'interprète).
Si des erreurs de lecture se produisent tout de même, le message suivant apparait :

** READ error (in LIBRARY) : <n>

dans lequel <n> est le numéro d'erreur codé d'un manière identique à l'erreur ** READ erreur : <n>.
D'autre part si une fin de fichier apparait dans un fichier LIBRARY alors qu'une S-expression était en cours de lecture, le message suivant apparait :

** E.O.F. during READ (in LIBRARY).

Dans l'état actuel du système, il ne peut y avoir qu'un fichier LIBRARY ouvert à la fois. Si donc un fichier LIBRARY appelle d'autres fichiers LIBRARY, l'erreur suivante apparait :

** LIBRARY error.

(LIBRARY <file>) [FSUBR]

charge le fichier <file> en silence (sans aucune impression). Ce fichier doit posséder l'une des extensions standards VLI VLA ou VLO. Si le fichier existe et si le chargement s'est correctement effectué, cette fonction ramène <file> en valeur i.e. le nom du fichier chargé. Danc tous les autres cas LIBRARY ramène NIL. Le répertoire disque dans lequel le fichier doit être cherché est précisé par la fonction suivante.

```
ex : (OR (EQ (TYPEFN 'PHENARETES) 'EXPR)
         (LIBRARY PHENAR)
         (ERROR "mais où est-elle donc passée ?"))
```

(PATHLIBRARY <ppn1> ... <ppnN>) [FSUBR]

indique à la fonction LIBRARY les différents répertoires dans lesquels elle doit rechercher le fichier. Le répertoire utilisateur est indiqué par NIL. La recherche s'effectue dans l'ordre des arguments donnés à cette fonction.
Par défaut (PATHLIBRARY () SYS), indique qu'il faut d'abord essayer de chercher le fichier dans le répertoire de l'utilisateur et en cas d'echec dans celui du système.

```
ex : (PATHLIBRARY
        ()              ; ppn de l'utilisateur ;
        SYS             ; ppn SYS: ;
        (LIS . JER)     ; ppn style IRCAM ;
        \(730 . 272))   ; ppn style TOPS10 ;
```

7.4 Les Fonctions De Sortie De Base

Toutes éditent dans un buffer de sortie totalement accessible à l'utilisateur et écrivent sur le fichier de sortie sélecté au moyen des fonctions OUTPUT ou FILOP. Si au cours d'une édition le buffer de sortie est plein, il est automatiquement imprimé, en utilisant la fonction (TERPRI 1), et l'édition se poursuit sur la ligne suivante.

(PRINT <s1> ... <sN>) [SUBR à N arguments]

édite dans le buffer de sortie les différentes S-expressions <s1> ... <sN> puis imprime ce buffer (après insertion des caractères Carriage Return / Line Feed). PRINT ramène <sN> en valeur.

(PRIN1 <s1> ... <sN>) [SUBR à N arguments]

Edite dans le buffer de sortie les différentes S-expressions <s1> ... <sN> sans imprimer le buffer. PRIN1 ramène <sN> en valeur.

(TERPRI <n>) [SUBR à 1 argument] {pour TERminate PRInt}

imprime le buffer de sortie, se positionne en début de ligne en imprimant le code Return, puis saute <n> lignes en imprimant <n> fois le code Line-feed. Si l'argument <n> n'est pas un nombre ou n'est pas fourni, TERPRI agit comme si <n> était égal à 1 provoquant un interlignage simple. L'appel (TERPRI) est donc équivalent à (TERPRI 1).
Si <n> = 0, le positionnement en début de ligne est quand même réalisé mais aucune ligne n'est sautée ce qui permet de réaliser une

surimpression .
Si <n> = -1, le positionnement en début de ligne n'a pas lieu et aucune ligne n'est sautée. Cette option permet d'imprimer des messages et d'attendre des réponses sur la même ligne ce qui facilite les dialogues.

```
ex : ? (DE QAM (msg) (PRIN1 msg) (TERPRI -1) (READ))
     = QUAM

     ? (QUAM "Nb de satellites de VEGA")
       Nb de satellites de VEGA ? xxx
                    (xxx est la réponse entrée)
     = xxx
```

(PAGE) [SUBR à 0 argument]

imprime le buffer de sortie puis saute une page (en envoyant le code Form-Feed). PAGE ramène NIL en valeur.

(PRINC <c> <n>) [SUBR à 2 arguments] {pour PRINt Character}

édite <n> fois le caractère <c>. Si <n> n'est pas un nombre ou est omis, le caractère <c> n'est édité qu'une fois. PRINC ramène <c> en valeur.

(SPACES <n>) [SUBR à 1 argument]

édite <n> fois le caractère espace. Si <n> n'est pas un nombre ou est omis, un seul espace est édité. SPACES ramène <n> en valeur.

(TAB <n>) [SUBR à 1 argument]

édite <n> fois le caractère tabulation. Si <n> n'est pas un nombre ou est omis, une seule tabulation est éditée. Les tabulations sont d'ordinaire en position 8, 16, 24, 32, 40 ... .

(TTAB <n>) [SUBR à 1 argument]

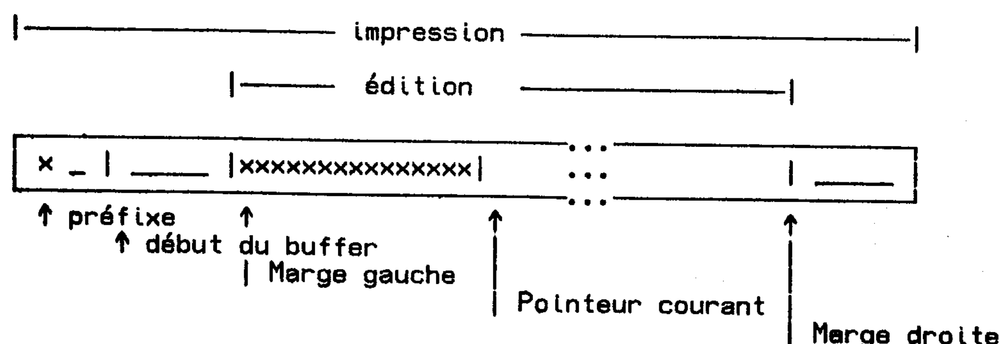
édite un nombre d'espaces tel que le nombre de caractères actuellement édités dans la ligne soit égal à <n>. Autrement dit (TTAB n) se met en position <n> du buffer de sortie. TTAB est utilisé pour réaliser des colonnages.

7.5 Contrôle Des Fonctions De Sortie

Les fonctions de sortie éditent les représentations externes des objets VLISP dans un buffer de sortie accessible au moyen de fonctions spécialisées. Un préfixe imprimable est ajouté au début de chacune des lignes imprimées.

7.5.1 Le Buffer De Sortie -

Le buffer de sortie est organisé de la maniére suivante :



Les éditions ne s'effectuent dans le buffer de sortie qu'entre la marge gauche et la marge droite alors que les impressions vont porter sur tout le buffer. L'accès aux valeurs de la marge gauche, du pointeur courant et de la marge droite est réalisé au moyen de fonctions STATUS ; il existe de plus une fonction OUTBUF permettant de manipuler directement n'importe quel caractère de ce buffer de sortie.

(STATUS 7 <n>)

permet de modifier (si <n> est donné et est numérique) ou de connaître la valeur de la marge gauche courante.

(STATUS 8 <n>)

permet de modifier (si <n> est donné et est numérique) ou de connaître la valeur de la position courante du pointeur sur le buffer de sortie. Ce pointeur pointe continuellement sur le premier emplacement disponible du buffer.

(STATUS 9 <n>)

permet de modifier (si <n> est donné et est numérique) ou de connaître la valeur de la marge droite.

(OUTBUF <n> <c>) [SUBR à 2 arguments] {pour OUTput BUFfer}

permet de modifier le <n>ième caractère du buffer de sortie avec le caractère <c>, si l'argument <c> est donné. OUTBUF ramène toujours la valeur du <n>ième caractère du buffer de sortie.

ex : Si les fonctions SPACES et TTAB n'étaient pas standard, on pourrait les définir de la manière suivante :

```
(DE SPACES (N)
   (IF (LT (PLUS (STATUS 8 N)) (STATUS 9))
       (STATUS 8 (PLUS (STATUS 8) N))
       (TERPRI)))

(DE TTAB (N)
   (AND (GEZP N) (LT N (STATUS 9)) (STATUS 8 N)))
```

7.5.2 Limitations D'impression -

Pour pouvoir limiter les impressions des structures très longues et rendre possible celles des listes circulaires deux nouvelles fonctions ont été ajoutées.

(PRINTLENGTH <n>) [SUBR à 1 argument]

permet de connaître et/ou de modifier (si l'argument numerique <n> est fourni) le nombre maximum d'éléments de liste qui est imprimé lors d'un PRINT ou d'un PRIN1. Par défaut la valeur de ce nombre est de 2000. Si une liste a un nombre d'éléments plus élevé, le reste des éléments ne sera pas imprimé et des points de suspension apparaitront devant la parenthèse fermante. Cette fonction permet d'imprimer des listes circulaires sur les CDR. PRINTLENGTH ramène en valeur la longeur maximum courante.

```
ex : (PRINTLENGTH 6)       ☞ 6
     '(1 2 3 4 5 6 7 8 9) ☞ (1 2 3 4 5 6...)
```

(PRINTLEVEL <n>) [SUBR à 1 argument]

permet de connaître et/ou de modifier (si l'argument numérique <n> est fourni) la profondeur maximum d'impression (i.e. le nombre maximum de parenthèses ouvrantes non fermées). En cas de dépassement, la liste qui provoque le débordement n'est pas imprimée et le caractère & est imprimé à sa place. Cette fonction permet d'imprimer des listes circulaires sur les CAR. PRINTLEVEL ramène en valeur la profondeur maximum courante.

```
ex : (PRINTLEVEL 3) ☞ 3

     '(DE FOO (L)
          (IF (NULL (CDR L)) L
              (FOO (CDR L))))
     ☞  (DE FOO (L) (IF (NULL &) L (FOO &)))

; impression des listes circulaires ;

?  (PRINTLEVEL 10)
=  10
=  ; time = 0 ms ;
?  (PRINTLENGTH 50)
=  50
=  ; time = 0 ms ;
?  (SETQ L '(X Y Z))
=  (X Y Z)
=  ; time = 0 ms ;
?  (RPLACD (CDDR L) L)
=  (Z X Y Z X Y Z X Y Z X Y Z X Y Z X Y Z X Y Z X Y Z X
=   Y Z X Y Z X Y Z X Y Z X Y Z X Y Z X Y Z X Y Z X...)
=  ; time = 0 ms ;
?  (RPLACA L L)
=  ((((((((((& Y Z & Y Z & Y Z & Y Z & Y Z & Y Z & Y Z &
=  Y Z & Y Z & Y Z & Y Z & Y Z & Y Z & Y Z & Y Z & Y Z &
=  Y...)...)...)...)...)...)...)...)...)...)
=  ; time = 0 ms ;
?
```

7.5.3 Les Préfixes -

A chaque impression du buffer de sortie un caractère préfixe est ajouté à gauche de la ligne, pour différencier les lignes imprimées par le TOPLEVEL, par l'utilisateur ...
Cette impression du préfixe est validée par le bit 28 du R.G. qui d'une manière standard est positionné.
Il existe 3 préfixes, tous redéfinissables :
- le préfixe indiquant une forme lue par la fonction TOPLEVEL (i.e. une forme à évaluer). Par défaut ce préfixe est le caractère ? .
- le préfixe indiquant la valeur retournée par une évaluation au TOPLEVEL. Par défaut ce préfixe est le caractère = .
- le préfixe indiquant des impressions ordonnées par des fonctions de l'utilisateur. Par défaut ce préfixe est le caractère espace. Il n'y a pas alors de préfixe visible.

Trois fonctions STATUS permettent de consulter et/ou de modifier ces préfixes; chacune de ces fonctions peut avoir un argument <c> qui doit être un caractère (i.e. un atome mono-caractère). Si cet argument est donné, il devient le nouveau caractère préfixe, s'il n'est pas donné, ces fonctions ramènent le caractère préfixe courant.

(STATUS 11 <c>) donne accès au préfixe de lecture au TOPLEVEL. Ce caractère est également envoyé sur le terminal avant toute lecture de ligne sur celui-ci.

(STATUS 12 <c>) donne accès au préfixe des résultats au TOPLEVEL

(STATUS 13 <c>) donne accès au préfixe des impressions de l'utilisateur.

```
ex :  ? (DE FOO () (PRINT 'NABUCHO) 'OK)
      = FOO

      ? (FOO)           ←   ligne lue par le TOPLEVEL
        NABUCHO         ←   impression de l'utilisateur
      = OK              ←   impression du résultat du TOPLEVEL
```

7.5.4 édition Standard -

Les différents objets VLISP sont édités dans le buffer de sortie en respectant l'unique régle suivante : la représentation externe d'un objet atomique ne peut être éditée sur deux lignes; ceci pour les atomes litéraux les nombres ou les chaînes de caractères.

L'impression d'un objet non VLISP (du genre adresse de lancement de SUBR ...) est faite en octal, préfixée par le caractère \ (qui est le macro-caractère d'entrée de passage en mode octal).
ex : (VAG 'CAR) ☞ \14000000

L'impression de la fonction (QUOTE <s>) s'effectue '<s> (donc en restituant le macro-caractère standard d'entrée) pour des raisons de lisibilité.

Des bits du R.G. permettent de régler certaines modalités d'impression en particulier :

- le bit 20 du R.G. indique que les impressions physiques doivent avoir lieu à la fin de chacun des remplissages du buffer VLISP et non à la fin du remplissage du buffer système associé au périphérique de sortie utilisé. Ce bit est positionné par défaut si le périphérique de sortie n'est pas un

terminal (TTY).

- le bit 21 du R.G. indique que toutes les éditions débutent par un espace. Ce bit est positionné par défaut.

- le bit 25 du R.G. indique qu'il faut insérer un espace entre chaque atome. Ce bit est également positionné par défaut.

Il existe d'autres indicateurs spécifiques à certains types d'objets :

- le bit 24 du R.G. indique que les caractères spéciaux des P-names des atomes litéraux doivent être précédés du caractère / (qui est le caractère "quote caractère"). Cette posibilité permet d'imprimer les représentation externes des atomes litéraux d'une manière telle qu'elles pourront être ensuite relues par la fonction d'entrée standard READ. Ce bit n'est pas positionné par défaut.

- le bit 27 du R.G. indique que les chaînes de caractères doivent être éditées encadrées du caractère délimiteur de chaîne " . Ce bit est positionné par défaut.

- le bit 22 du R.G. indique que les nombres positifs doivent débuter par le signe + . Ce bit n'est pas positonné par défaut.

- le bit 23 du R.G. indique que les nombres négatifs doivent débuter par le signe - . Ce bit est positionné par défaut.

7.5.5 Macro Fonction De Sortie : MACOUT -

Si au cours d'une édition les fonctions PRINT ou PRIN1 doivent éditer une liste dont le CAR, atomique, contient sur sa P-liste l'indicateur MACOUT , la fonction associée à cet indicateur est lancée avec pour argument le CDR de la liste qui devait être éditée. Ces fonctions sont appellées des Macros fonctions de sortie, et sont définies au moyen de la fonction DMO.
Il existe un indicateur, le bit 26 du R.G. qui permet de valider ce lancement : si ce bit est positionné (ce qui est l'option par défaut) les MACOUT sont traitées. Si ce bit n'est pas positionné, les appels des MACOUT sont édités sans modifications.

(DMO <at> <l> <s1> ... <sN>) [FSUBR]
{pour Define Macro Output}

définit une macro fonction de sortie associée à l'atome littéral <at>, la liste des arguments de cette fonction est <l> et le corps de la fonction est <s1> ... <sN>.
DMO ramène <at> en valeur et pourrait être définie en VLISP :

```
(DF DMO (l)
    (PUT (CAR l) (CONS 'LAMBDA (CDR l)) 'MACOUT))
```

Il existe un certain nombre de macro fonctions de sortie prédéfinies dans le système, elles correspondent aux macro caractères standard.
(QUOTE <objet>) est restitué '<objet>
(LIST <obj1> ... <objN>) est restitué [<obj1> ... <objN>]

Si le restitution du macro caractère ' n'était pas standard on pourrait le définir :

```
(DMO QUOTE (X) (PRIN1 '/' (CAR X)))
```

7.6 Fonctions Spéciales Sur TTY

Un ensemble de fonctions permettent de travailler directement sur le terminal (TTY) pour utiliser au mieux toutes ses possibilités. Ces fonctions n'utilisent pas le système de fichier classique et peuvent donc être utilisées à tout moment.

(TYI) [SUBR à 0 argument] {pour TtY Input}

ramène un nombre représentant le code interne du caractère suivant lu sur la TTY, sans aucune conversion et sans utiliser l'éditeur de ligne du moniteur (en particulier les caractères RUB-OUT, et les différents control-caractères ne sont plus effectifs). Dans la version IRCAM du système, les caractères lus par cette fonction sont représentés sur 9 bits en utilisant les terminaux DM.

(TYS) [SUBR à 0 argument] {pour TtY Sneak}

si un caractère est prêt à être lu sur le terminal, ramène la valeur que ramenerait la fonction TYI, sinon si aucun caractère n'est prêt TYS ramène NIL. Cette fonction permet de tester si un caractère a été frappé sur le terminal. ATTENTION : le caractère n'est pas reellement lu mais simplement consulté (cf: la fonction PEEKCH). Donc (UNTIL (TYS) (TYI)) est équivalent à (TYI).

(TYO <n>) [SUBR à 1 argument] {pour TtY Output}

imprime sur le terminal le caractère de code interne <n>. Il n'est hélas pas possible d'envoyer avec cette fonction les caractères spéciaux de positionnement des terminaux DM; la fonction DISPLAY réalisera les transmissions spéciales.

(DISPLAY <l> <n>) [SUBR à 2 arguments]

<l> est une liste de codes internes. DISPLAY envoie sur le terminal la liste de ces codes internes. Pour cette fonction, les codes internes peuvent être des commandes spéciales au terminal. Chaque code de commande doit être précédé du caractère \177 (valeur 177 octale).
Le deuxième argument <n> s'il est donné sert à spécifier les autres indicateurs spéciaux de l'UUO UPGIOT et n'est donc pratiquement jamais utilisé.
L'utilisation de cette fonction étant très délicate, un certain nombre de fonctions standard AUTOLOAD de display ont été regroupées dans le fichier SYS:DISPLAY.VLI qui permettent, en outre, d'avoir un contrôle absolu sur l'écran durant tout le programme de visualisation.
Ce fichier est donné en Appendice B.

Liste des principaux codes des terminaux DM

code octal	effet
\002	HOME. positionne le curseur en haut et à gauche de l'écran.
\007	BELL. fait sonner la cloche du terminal
\010	BACK CURSOR. fait reculer le curseur d'une position à gauche.
\011	TAB.

effectue une tabulation à droite.
\012 LINEFEED.
descend d'une ligne.
\013 TAB CLEAR.
efface la tabulation à cet endroit.
\014 SET CURSOR POSITION.
les deux codes suivants doivent être les nouvelles positions en X et en Y du curseur, ces valeurs devant être XORées avec la valeur \140.
\015 CARRIAGE RETURN
revient en début de ligne.
\031 SET TAB.
positionne une tabulation à cet endroit.
\032 UP CURSOR.
remonte le curseur d'une ligne.
\034 FORWARD CURSOR.
avance le curseur d'une position à droite.
\036 MASTER CLEAR.
efface l'ecran, les tabulations et tous les autres modes.

exemples de fonctions utilisant les terminaux DM

```
(DE BEEP ()
   ; fait sonner la sirène ;
   (DISPLAY '(\177 \007))

(DE OUTXYS (X Y S)
   ; édite à la Xième ligne, Yième colonne la chaîne S ;
   (DISPLAY
      (APPEND
        [\177 \14 (LOGXOR \140 Y) (LOGXOR \140 X)]
        (MAPCAR (MAKLIST S) 'CASCII))))
```

(PPIOT <n1> <n2>) [SUBR à 2 arguments]

appelle l'UUO IRCAM PPIOT avec <n1> comme numéro de fonction et <n2> comme argument. Cette fonction permet d'utiliser les "page of paper" des écrans en DM mode. Tous les renseignements utiles d'utilisaton de cette fonction se trouvent dans la brochure "The IRCAM/STANFORD Monitor" by Martin FROST et Brian HARVEY. La principale fonction est : (PPIOT 0 n) qui permet de sélecter la page de papier de numéro n.
Certaines fonctions du fichier SYS:DISPLAY.VLI utilisent cette fonction (cf: Appendice B).

(TRMOP <n1> <n2> <n3>) [SUBR à 3 arguments]

appelle l'UUO TRMOP qui permet de connaître et/ou de modifier les caractéristiques du terminal.
<n1> est le numéro de la fonction à exécuter
<n2> est le numéro UDX du terminal
<n3> est la valeur de modification.

Toutes ces fonctions sont décrites dans le MONITOR CALL standard de DEC certains autres ne le sont que dans le STANFORD/IRCAM MONITOR.

par exemple pour savoir si un terminal est un DATAMEDIA :

```
(LZP (TRMOP \1043)))
```

pour supprimer la brillance des lignes activées par l'éditeur :

(TRMOP \1042 () 1)

7.7 Les Spécifications De Fichiers

Une spécification de fichier a, en VLISP 10, la forme suivante :

(<dev> (<file> . <ext>) (<proj> . <prog>) <prot>)

dans laquelle :

```
<dev>     : est le nom du périphérique (6 caractères au maximum).
<file>    : est le nom du fichier (6 caractères au maximum).
<ext>     : est le nom de l'extension (3 caractères au maximum).
<proj>    : est le numéro (ou nom) de projet.
<prog>    : est le numéro (ou nom) du programmeur.
<prot>    : est le numéro de la protection.
```

L'ensemble (<proj> . <prog>) est la spécification d'un répertoire <ppn> . L'absence de cette spécification implique l'utilisation du répertoire de l'utilisateur. Les répertoires de type IRCAM (i.e. pouvant contenir des lettres) sont correctement traités.

spécifications par défaut :

```
device    : TTY (le terminal)
filename  : en entrée LISPIN, en sortie LISPOU
extension : en entrée VLI pour un fichier écrit en VLISP.
                      VLA ou VLO pour des fichiers écris en LAP.
            en sortie LST pour les sorties normales.
                      VLP pour les sorties issues du PRETTY-PRINT.
                      VLC pour les sorties issues du CROSS.
                      VLX pour les sorties issues de l'INDEX.
projet.programmeur : celui de l'utilisateur
protection : \055
```

Par défaut donc,

```
(INPUT)          == (INPUT '(TTY (LISPIN . VLI) () ))
(OUTPUT)         == (OUTPUT '(TTY (LISPOU . LST) () \055 ))
```

Une nouvelle écriture allégée est également possible, constituée d'un atome littéral <at>; elle correspond à la forme :

```
        (DSK (<at> . extension standard) ())
(INPUT 'FOO)     == (INPUT '(DSK (FOO . VLI) () ))
(OUTPUT 'FOO)    == (OUTPUT '(DSK (FOO . LST) () ))
```

Il est possible dans certaines fonctions standard de type EXPR, contenues dans le fichier initial SYS:VLISP.INI (par exemple la fonction FILE cf: l'appendice A) d'utiliser des chaînes de caractères pour spécifier un fichier. Cette chaîne de caractères représente la spécification d'un fichier dans le style DEC i.e. :

<dev>:<filename>.<ext>[<prg>,<prj>]<<pro>>

ex : "DSK:FILE.TXT[LIS,JER]<055>" correspond à
(DSK (FILE . TXT) (LIS . JER) \055)

7.8 La Sélection Des Fichiers D'entrée/sortie

Dans l'état actuel du système, VLISP 10 peut gérer simultanément
- un fichier d'entrée
- un fichier de sortie
- un fichier library
- un fichier auxiliaire
- un terminal

Un système multi-fichiers est en cours de réalisation.

(INPUT <file>) [SUBR à 1 argument]

cette fonction permet de sélecter le fichier d'entrée <file>, après avoir fermé le fichier d'entrée précédent. A l'ouverture d'un fichier TTY, le message --- ALLO ? --- est imprimé sur cette TTY, un ? est imprimé au début de chaque ligne et un petit Pretty-print est activé; ce Pretty-print, qui tient compte du nombre de parenthèses non encore refermées, effectue des indentations automatiques en début de chaque ligne. Un indicateur, le bit 11 du R.G. , est positionné dans ce cas et peut être testé pour savoir si le fichier d'entrée est bien un terminal.

Les erreurs suivantes peuvent apparaitre :

** open error (input).

Le périphérique d'entrée est incorrect (inexistant ou impossible à ouvrir en lecture). Le fichier d'entrée standard est ouvert à sa place.

** lookup error (input) : <n>

Le fichier d'entrée est incorrect. Le fichier d'entrée standard est utilisé à sa place. <n> est le type de l'erreur. Parmi les plus fréquentes signalons :
- 0 le fichier n'existe pas
- 1 le <ppn> est incorrect
- 2 le fichier est trop protégé
- 6 erreur physique de lecture

(EOF) [SUBR à 0 argument] {pour End Of File}

cette fonction est automatiquement lancée par le système à la fin du fichier d'entrée. EOF peut être également lancé par l'utilisateur pour provoquer artificiellement des fins de fichiers en lecture (le "SIGNAL EOF;" cher aux PL1istes). D'une manière standard EOF imprime le message d'avertissement ** E.O.F puis re-ouvre le fichier standard d'entrée. Toutefois si une S-expression était en cours de lecture (non terminée) le message devient ** E.O.F. during READ. Ce message indique en général qu'il manque des parenthèses fermantes à la fin du fichier. EOF peut bien évidemment être redéfinie sous forme d'une EXPR pour gérer soi-même les fins de fichiers.

L'action standard de cette fonction peut être décrite en VLISP :

```
(DE EOF ()
   (PRINT '**E.O.F)
   (INPUT))
```

Si cette fonction est redéfinie par l'utilisateur, elle doit comporter explicitement une ouverture d'un fichier d'entrée (au moyen de la fonction INPUT) sous peine de déclencher une erreur moniteur :

?IO TO UNASSIGNED CHANNEL AT USER PC xxx

(OUTPUT <file>) [SUBR à 1 argument]

sélecte le nouveau fichier de sortie <file> après avoir fermé le fichier de sortie précédent.

Les erreurs suivantes peuvent apparaitre :

** open error (output).

Le périphérique de sortie est incorrect (inexistant ou impossible à ouvrir en écriture). Le fichier de sortie standard est ouvert à sa place.

** enter error (output) : <n>

Le fichier de sortie est incorrect. Le fichier de sortie standard est utilisé à sa place. <n> est le type de l'erreur. Parmi les plus fréquentes signalons :
1 le <ppn> est incorrect
2 le fichier est trop protégé
6 erreur physique d'écriture
14 il n'y a plus de place sur le disque

(FILOP <l> <file1> <file2>) [SUBR à 3 arguments]
{pour FILe OPeration}

permet de réaliser la plupart des fonctions sur les fichiers (DELETE, RENAME, APPEND ...). FILOP utilise la nouvelle UUO FILOP . <l> est une liste de la forme [no-de-canal no-de-fonction], <file1> est la spécification du fichier à utiliser, <file2> est la spécification d'une deuxième fichier qui n'est utilisé que pour la fonction RENAME. L'emploi de cette fonction étant malaisé et parfois dangereux, il est préférable d'utiliser la fonction FILE , présente dans le fichier SYS:VLISP.INI (voir l'annexe A), qui assure un contrôle plus sérieux et offre une syntaxe plus agréable.

Toutefois pour pouvoir utiliser cette fonction les renseignements suivant sont nécéssaires :

numéro des canaux
1 canal d'entrée (INPUT)
2 canal de sortie (OUTPUT)
3 canal library (LIBRARY)

les principaux codes fonction sont :
1 lecture d'un fichier
2 création d'un nouveau fichier
3 écriture d'un fichier
6 extension d'un fichier
7 fermeture d'un fichier
8 ckeckpoint d'un fichier
11 changement de nom d'un fichier
12 destruction d'un fichier
13 préallocation d'un fichier

FILOP ramène NIL en valeur si la fonction sur fichier s'est correctement déroulée et ramène le numéro du code de l'erreur dans le cas contraire.
Les principaux codes d'erreur sont :
0 fichier inexistant
1 répertoire incorrect
2 protection trop forte
3 fichier en cours de modification
4 fichier existant
6 erreur d'entrée/sortie

ex : de manipulation de fichiers

```
; fonction COPYFILE qui copie entièrement le fichier
  d'entrée <filin> dans le fichier de sortie <filout> ;

  (DE COPYFILE (<filin> <filout>)
      (INPUT <filin>)
      (OUTPUT <filout>)
      (WITH (EOF () (finfich))
         (ESCLOOP finfich
            (PRINT (READ))))
      (OUTPUT)
      (INPUT)
      'VOILA))
```

7.9 Autres Fonctions Sur Les Fichiers

7.9.1 Lancement D'autres Programmes -

(RUN <file> <n>) [SUBR à 2 arguments]

permet de sortir de VLISP et de jouer le programme de nom <file>. Si le 2ème argument <n> est fourni, il fait office de déplacement (offset) pour l'adresse de lancement du nouveau programme (si celui-ci a été prévu à cet effet).

ATTENTION : RUN effectue un aller-simple!

ex : (RUN '(SYS (E . SHR))) va jouer E

(RUN '(SYS (E . SHR)) -1) va jouer E en utilisant le même fichier qu'au dernier appel de E

(RUN '(SYS (ADV . EXE))) vous voici à l'aventure

7.9.2 Fonctions Sur Les <ppn> -

(GETPPN) [SUBR à 0 argument]

ramène le répertoire courant de l'utilisateur.

ex : (GETPPN) ☞ (LIS . DOC)

(ALIAS <ppn>) [SUBR à 1 argument]

permet de changer le nom du répertoire courant de l'utilisateur. Si aucun argument n'est fourni, le répertoire du LOGIN est utilisé. Cette fonction n'est effective que dans la version IRCAM et correspond à la comande moniteur .ALIAS ppn .

```
ex : (GETPPN)                  ☞ (LIS . P)
     (ALIAS '(LIS . DOC))      ☞ (LIS . DOC)
     (GETPPN)                  ☞ (LIS . DOC)
     (ALIAS)                   ☞ (LIS . P)
```

(DIRECTORY <ppn> <l>) [SUBR à 2 arguments]

ramène la liste de tous les fichiers du répertoire <ppn>. Cette fonction est utilisée pour s'assurer de la présence de fichier en restant sous VLISP. Le deuxième argument <l> sert à préciser le filtre ("wild-card") à utiliser pour isoler un ensemble de fichiers dont on précise le nom et/ou l'extension. Ces filtres permettent, dans le cas où l'on précise un nom et une extension, d'isoler un fichier unique ; DIRECTORY peut donc être utilisé pour tester la présence d'un fichier dans un répertoire.

Pour tester si le fichier (FOO . VLI) existe dans le répertoire (1 . 4), il suffit d'évaluer :
    (DIRECTORY '(1 . 4) '(FOO . VLI))
qui ramènera (FOO . VLI) ou NIL.

```
ex : (DIRECTORY)   ☞ ((FOO . VLI) (FOO . LST) (FOO2 . VLI))
     (DIRECTORY () '( () . VLI))  ☞ ((FOO . VLI) (FOO2 . VLI))
     (DIRECTORY () '(FOO))        ☞ ((FOO . VLI) (FOO . LST))
```

(SHOWIT <n>) [SUBR à 1 argument]

permet d'afficher sur la 3eme "who line" des écrans en DM mode dans le système IRCAM, tous les renseignements relatifs au fichier ouvert sur le canal <n>. Ce numéro de canal vaut 1 pour le fichier d'entrée, 2 pour le fichier de sortie et 3 pour le fichier library.

7.9.3 Les Fichiers TMPCOR -

Il est possible d'utiliser les fichiers TMPCOR sous VLISP. La seule fonction disponible actuellement est la lecture d'un tel fichier.

(TMPCOR <at>) [SUBR à 1 argument]

ramène la chaîne de caractères correspondant au fichier TMPCOR de nom <at>. Si un tel fichier n'existe pas, TMPCOR ramène NIL. Cette fonction est en outre utilisée pour récupérer le TMPCOR des éditeurs pour connaître le dernier fichier édité, ou par la suite pour créer de nouveaux TMPCOR pour VLISP ou pour les autres programmes systèmes.

le nom des éditeurs à l'IRCAM est

```
ED      pour ETV
EDS     pour TECO
VLI     pour VLISP
```

on peut donc lire le TMPCOR de ETV par l'appel :

(TMPCOR 'ED) ☞ (E T F I L I O . R N M (1 9 P 2 4 L 1 9 , 2 4 M))

7.9.4 Les Fichiers Image-mémoire -

Deux nouvelles fonctions ont été crées pour pouvoir sauver l'image mémoire de votre zone de travail sur disque. Ces deux nouvelles fonctions deviendront par la suite inutiles à la mise en service de la nouvelle UUO IRCAM SWAP.
Il existe actuellement une autre méthode qui permet de stocker une image de tout l'interprète sur disque (voir les "démarrages à chaud" dans le chapitre suivant).
Pour ces deux fonctions, si vous ne précisez pas de fichier,
(DSK (TEMPOR . COR) ()) .
sera utilisé par défaut.

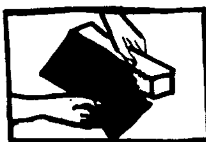
(WRCORE <file>) [SUBR à 1 argument]

écrit dans le fichier spécifié <file> votre image memoire. Cette fonction après exécution retourne au top-level de VLISP.

(RDCORE <file>) [SUBR à 1 argument]

restaure votre image mémoire à partir du fichier spécifié. Comme WRCORE cette fonction retourne après exécution au top-level de VLISP.
ATTENTION : l'interprète qui exécute cette fonction doit avoir la même configuration que l'interprète qui a crée ce fichier.





CHAPITRE 8

ORGANISATION ET UTILISATION

L'interprète se trouve dans le segment haut (HIGHSEG) de la mémoire. Sa taille actuelle est de 8 k (environ). Il est partageable et réentrant. Les vertus d'une telle organisation sont multiples, elle permet entre autre de n'avoir qu'une copie de l'interprète en mémoire dans le cas de plusieurs utilisateurs et d'accélerer les swapping du système qui, dans la mesure du possible, laissera la partie haute de l'interprète en mémoire.

8.1 Le Découpage De La Mémoire

La mémoire est divisée en zones fixes (dont les tailles peuvent être changées à l'initialisation du système). Il y a 8 zones dans le segment bas (LOWSEG) et une zone dans le segment haut (HIGHSEG).

Zone 0 du segment bas :

cette zone contient les mémoires de travail de l'interprète (buffers, zones de sauvetages diverses ...) ainsi que certaines fonctions flottantes. La taille de cette zone varie entre 1 et 2 k en fonction du nombre de fonctions flottantes chargées.

Zone 1 du segment bas :

c'est dans cette zone que sont stockés les atomes litéraux a raison de 6 mots par atome. La taille standard de cette zone permet de contenir 600 atomes. Il existe environ 400 atomes prédéfinis dans le système (les constantes et les fonctions standard), chaque utilisateur peut donc disposer d'environ 200 atomes litéraux pour ses besoins propres.

Zone 2 du segment bas :

c'est dans cette zone que sont stockés les nombres. Il exista 3 categories de nombres.
a) Les nombres entiers qui possèdent une représentation unique et sont stockés sur un seul mot (d'une manière standard tous les nombres compris entre -64 et +127 sont de ce type).
b) Les nombres entiers qui peuvent avoir plusieurs représentations simultanées et qui sont stockés sur 2 mots.
c) Les nombres flottants qui peuvent avoir également plusieurs représentations simultanées et qui sont stockés sur 2 mots.
La taille standard de cette zone permet de stocker jusqu'à 1000 nombres de n'importe quel type.

Zone 3 du segment bas :

Cette zone contient les pointeurs de chaînes. Chacun de ces pointeurs utilise 1 mot mémoire. La taille standard de cette zone permet de stocker jusqu'à 100 pointeurs de chaînes.

Zone 4 du segment bas :

Cette zone contient les doublets de liste stockés à raison d'1 mot par doublet. La taille standard de cette zone permet de stocker 8000 doublets de liste.

Zone 5 du segment bas :

Cette zone contient la pile de l'interprète et a une taille standard de 1200 mots.

Zone 6 du segment bas :

c'est dans cette zone que sont stockés la pile utilisateur et les tableaux. Un élément de pile ou un élément de tableau occupe 1 mot mémoire. La taille standard de cette zone est de 500 mots.

Zone 7 du segment bas :

c'est dans cette zone qu'est chargé le code résultant du LAP ou du compilateur. Sa taille standard est de 200 mots.

Zone du segment haut :

Cette zone, partageable par plusieurs utilisateurs, contient l'interprète lui-même. Sa taille actuelle se situe aux alentours de 8 k et n'est bien évidement pas configurable.

8.2 Le Garbage-Collecting

Chacune de ces zones est allouée d'une manière spécifique. Lorsque l'une de ces zones vient à saturation, une machinerie connue sous le nom de Garbage-Collecting (G.C.) est invoquée, pour récupérer la place perdue. Chaque zone possède un dispositif de récupération qui lui est propre. Si cet essai s'avère infructueux (la zone ayant provoquée le G.C. reste saturée) un message d'erreur est imprimé. Ce type d'erreur est fatal ; il y a donc retour immédiat au top-level. Il faut recommencer le travail en augmentant la taille de la zone incriminée au moyen de la fonction CONFIGURATION.
Si vous interrompez l'interprète (en tapant un ↑C) durant un G.C., le message suivant apparait :
↑ G.C. ↑ TYPE .CONT PLEASE.
vous devez émettre la commande .CONT pour terminer le G.C. puis reinterrompre l'interprète. Si vous relancez l'interprète (par la commande .REE) qui n'a pas terminé un G.C., vous le placez dans un état de confusion épouvantable.

Quand le système commence à exécuter un G.C. il positionne un indicateur, le bit 35 du R.G. . Cet indicateur est remis à 0 en fin de G.C.

Messages relatifs au remplissage d'une zone mémoire.

```
** no room for atoms.   remplissage de la zone atome.
** no room for numbers. remplissage de la zone nombre.
** no room for strings. remplissage de la zone chaîne.
** no room for lists.   remplissage de la zone liste.
** no room for arrays.  remplissage de la zone tableau.
** no room for code.    remplissage de la zone code.
```

Il est possible d'avoir des statistiques à chaque "Garbage-Collecting" en positionnant le bit 5 du R.G. (plus précisement en évaluant la forme (STATUS 1 5)). Après chaque "Garbage-Collecting", le texte suivant sera imprimé :

```
**** G.C. No            : xx
     G.C. Count         : xx
     ALTERED CELLS      : xx
     Lists         size : xxxx  marked : xxxx  freed : xxxx
     Strings       size : xxxx  marked : xxxx  freed : xxxx
     Numbers       size : xxxx  marked : xxxx  freed : xxxx
     Atoms         size : xxxx  marked : xxxx  freed : xxxx
     Code          size : xxxx  marked : xxxx  freed : xxxx
     Elapsed time       : xxxx  milli-sec.
     Average time in G.C. : xx %
```

Ces statistiques permettent de connaître :
- le numéro du garbage-collecting (le nombre de G.C. effectués depuis le lancement de VLISP).
- le compte actuel de G.C. (utilisé par la fonction (STATUS 23))
- le nombre de doublets de listes irrécupérables. Si ce nombre n'est pas égal à 0, les G.C. suivant ne seront pas garantis ... Il vaut mieux recharger un nouvel interprète ...
- le nombre total d'atomes, utilisés et libres.
- le nombre total de nombres, utilisés et libres.
- le nombre total de chaînes, utilisées et libres.
- l'occupation de la zone code.
- le temps qu'a duré ce G.C. en millisecondes.
- le pourcentage de temps passé uniquement dans le G.C. par rapport au temps d'utilisation total de l'interprète.

Dans la version I.R.C.A.M. de VLISP 10, un résumé de ces statistiques est systématiquement affiché sur la 3ème "who line" des écrans fonctionnant en DM mode. Cette action est validée par le bit 6 du R.G. qui par défaut est positionné.
La forme de ce résumé est :

```
xx # xx % xxxx l xxxx at
```

et contient les renseignements suivants (dans l'ordre)
- le numero du G.C. (le nombre de G.C. effectués depuis le lancement de VLISP)
- le pourcentage de temps passé uniquement dans le G.C. par rapport au temps d'utilisation total de l'interprète.
- le nombre de doublets libres.
- le nombre d'atomes libres.

Certains STATUS permettent de contrôler le fonctionnement du G.C.

(STATUS 21)

lance un G.C. et ramène en valeur le nombre de doublets libérés.

(STATUS 22)

ramène le nombre de doublets libres (mais sans lancer de G.C.).

(STATUS 23 <n>)

donne une valeur <n> au nombre de G.C. que l'interprète pourra effectuer avant de provoquer une erreur dont le libellé est :
** ER G.C. STEP DONE.

(STATUS 24 <n>)

donne un nombre minimum <n> de doublets récupérables à chaque G.C. Si ce nombre n'est pas atteint, il y a impression du message d'avertissement suivant :
** LEFT cells : <n>
où <n> est le nombre de doublets réellement récupérés.

8.3 Les Changements De Configuration

Il est possible, en début de travail, de changer la taille des zones standard. La fonction CONFIGURATION exécute ce travail. Cette fonction, si elle est utilisée, doit être la PREMIERE et DERNIERE forme à évaluer sur le fichier initial CONFIG.INI .

(CONFIGURATION init in out atoms nbs strgs lists stack array code)
[SUBR à 10 arguments]

cette fonction possède 10 arguments qui sont dans l'ordre :
- init : spécification du nouveau fichier initial
- in : spécification du fichier standard d'entrée
- out : spécification du fichier standard de sortie
- atoms : nombre total d'atomes.
- nbs : nombre total de nombres.
- strgs : nombre total de chaînes.
- lists : nombre total de doublets.
- stack : taille de la pile de l'interprète.
- array : taille de la zone des tableaux.
- code : taille de la zone de stockage du code.

Ces arguments doivent être donnés DANS L'ORDRE . Le 1er argument est obligatoire et ne doit pas être '(DSK (CONFIG . INI)) sous peine de faire boucler l'interprète durant le prélude. Si on ne désire pas modifier la taille d'une zone, l'argument à fournir est NIL. S'il n'est pas possible d'allouer la mémoire demandée, le message suivant est imprimé :
** not enough core.
et la configuration n'est pas modifiée.

exemples d'utilisation de la fonction de configuration.

```
; définition de la configuration standard ;
; cette configuration occupe :
     - 18 k en LOWSEG
     -  8 k en HIGSEG
 soit 26 k au total   ;

 (CONFIGURATION
```

```
        '(SYS (VLISP . INI))      ; fichier initial ;
        '(TTY (VLISP . VLI))      ; fichier d'entrée TTY ;
        '(TTY (VLISP . LST))      ; fichier de sortie TTY ;
        600                       ; atomes littéraux ;
        1000                      ; nombres;
        100                       ; chaînes ;
        8000                      ; listes ;
        1200                      ; pile système ;
        500                       ; tableaux ;
        200                       ; code ;
)))))

; définition d'une petite configuration batch ;
; cette configuration occupe :
     - 12 k en LOWSEG
     -  8 k en HIGSEG
soit   20 k au total    ;

(CONFIGURATION
        '(SYS (VLISP . INI))      ; fichier initial ;
        '(DSK (VLISP . VLI))      ; fichier disque d'entrée ;
        '(DSK (VLISP . LST))      ; fichier disque de sortie ;
        500                       ; 100 atomes utilisateur ;
        500                       ; 500 nombres ;
        50                        ; 50 chaînes ;
        4000                      ; 4000 doublets de liste ;
        1000                      ; pile de l'interprète ;
        0                         ; pas de tableaux ;
        0                         ; pas de code ;
)))
```

8.4 Lancement De L'interprète

L'interprète VLISP 10.3 se trouve dans la zone système. Pour l'utiliser il suffit d'émettre la commande :

.R VLISP ou bien .VLISP

Si l'erreur ?VLISP.SAV not found apparait, l'interprète ne se trouvait pas sur le disque système et il n'y a plus rien à faire.
Avant de commencer le dialogue sur les fichiers standards d'entrée-sortie, le fichier initial CONFIG.INI est chargé, s'il existe dans la partition de l'utilisateur. S'il n'existe pas le message ** LOOKUP ERROR (INPUT) : 0 est imprimé, et le fichier standard d'entrée est ouvert. Ce fichier initial peut contenir une reconfiguration des zones de l'interprete ou des définitions de fonctions préalables à un travail donné ou bien encore une définition de travaux à exécuter en "batch".
Il existe un fichier d'initialisation standard de nom (SYS (VLISP . INI)) qui contient la définition de macro-caractères, de fonctions synonymes, de fonctions autoloads, de fonctions utilisant des fichiers, des traps erreurs ... Ce fichier a le bon goût de lire après son chargement le fichier VLISP.INI qui se trouve sur votre partition.
Une pratique utile est de mettre à la fin de ce nouveau fichier, les commandes pour charger le dernier fichier manipulé par l'éditeur ETV ou TECO mais uniquement si ce fichier possède une extension .VLI).

L'ordre de chargement de ces fichiers est donc :

1) chargement s'il existe du fichier (DSK (CONFIG . INI)) qui se trouve sur votre partition.

2) si le 1er argument de la fonction CONFIGURATION du fichier précédent est (SYS (VLISP . INI)) , chargement de ce fichier

3) si le 2) a été chargé, chargement du fichier (DSK (VLISP . INI)) qui se trouve sur votre partition.

4) si le 3) a été prévu à cet effet, chargement du dernier fichier manipulé par l'éditeur si celui-ci possède une extension .VLI.

L'appendice A contient le listages des 3 fichiers :

- CONFIG.INI	(1)	standard (qui permet de lire (2))
- SYS:VLISP.INI	(2)	le fichier d'initialisation standard du système
- VLISP.INI	(3)	un fichier d'initialisation utilisateur qui permet de charger le dernier fichier modifié

Dans la version IRCAM de l'interpréte une possibilité de passage entre l'interprète et l'éditeur ETV est disponible :

pour passer de VLISP à l'éditeur il suffit d'invoquer
(RUN (SYS (E . SHR)) -1)
pour accéder immédiatement au dernier fichier édité (c'est donc l'équivalent de la comande moniteur .ETV sans argument).
Il existe dans le fichier standard SYS:VLISP.INI la définition d'un macro-caractère (le CONTROL-E) qui permet d'appeller l'éditeur. Il est défini :
(DMC é () (RUN (SYS (E . SHR)) -1))

pour passer de l'éditeur à VLISP, une nouvelle commande y a été ajoutée, la commande étendue : [EDIT] X VLISP (qui peut être abrégée [EDIT] X VL) Cette commande permet de rentrer dans l'interprète en positionnant le bit 29 du R.G. qui indique que l'on arrive directement de l'éditeur ETV. Cet indicateur est par la suite testé pour savoir s'il faut charger automatiquement le dernier fichier modifié par l'éditeur.

8.5 La Boucle TOPLEVEL

Une fois ce prélude terminé, l'interprète rentre dans la boucle principale qui consiste à évaluer indéfiniment la forme :
(WHILE T (TOPLEVEL))
C'est donc la fonction TOPLEVEL qui détermine le mode de fonctionnement de l'interprète. Cette fonction (de type SUBR) est bien évidemment redéfinissable par l'utilisateur qui désire se construire son propre système.

(TOPLEVEL) [SUBR à 0 argument]

Cette fonction va, d'une manière standard, :
- lire une S-expression sur le fichier d'entrée courant
- si le bit 2 du R.G. est positionné (ce qui est l'option par défaut), imprimer cette S-expression
- évaluer cette S-expression et stocker le résultat de cette évaluation dans l'atome IT
- si le bit 1 du R.G. est positionné (ce qui est l'option par défaut), imprimer le résultat de cette évaluation
- si le bit 0 du R.G. est positionné (ce qui est l'option par défaut), imprimer le temps qu'a duré cette évaluation. La fréquence de l'horloge du PDP 10 étant de 50 Hz, les durées de 0 et de 20 ms ne sont pas significatives.

Le trait IT permet, en mode conversationnel, d'effectuer des calculs à la chaîne :

```
? 4
= 4
? (* it 2)
= 8
? (* it it)
= 64
...
```

TOPLEVEL pourrait se définir en VLISP de la manière suivante :

```
(DE TOPLEVEL ( ;; lu evtime)
   (SETQ lu (READ))
   (IF (STATUS 4 2) (PRINT lu))
   (SETQ evtime (RUNTIME))
   (SETQ IT (EVAL lu))
   (SETQ evtime (DIFFER (RUNTIME) evtime))
   (IF (STATUS 4 1) (PRINT IT))
   (IF (STATUS 4 0) (PRINT "; time =" evtime " ms. ;")))
```

ATTENTION : une redéfinition de cette fonction, qui n'évalue pas des formes lues, rend le système inutilisable.

exemple de définition à éviter :

```
(DE TOPLEVEL () (READ) (PRINT 'MARRE))
```

8.6 Les Démarrages à Chaud

Il est possible de sauver à tout moment une image de l'interprète et de sa zone de travail en :
- interrompant l'interprète au moyen d'un ↑C quand celui-ci se trouve au niveau du TOP-LEVEL.
- en émettant la commande :
  .NSAV nnnnn xxK

qui sauve une image de la mémoire dans un fichier disque de nom nnnnn et d'extension par défaut .EXE , la mémoire sauvée a pour taille maximum xxK.

Le fichier ainsi créé peut être appellé de la même manière qu'un programme normal i.e. au moyen de la commande moniteur .RUN .
Ce programme contient l'interprète VLISP dans l'état où il se trouvait au moment de l'interruption; en particulier la taille des zones standard, toutes les définitions des fonctions ainsi que toutes les valeurs des variables sont conservées.
Ces fichiers sont utilisés pour effectuer des mises au point ou bien pour garder des versions prêtes (et en général compilées) de programmes utilisés fréquemment.

Attention : les fichiers ainsi créés sont très gros ; de l'ordre d'une ou de plusieurs centaines de blocs disques.

Lors du "démarrage à chaud", quand on relance l'interprète, la fonction HOT.START est utilisée en lieu et place de la fonction TOPLEVEL dans la boucle principale tant qu'un indicateur reste positionné.
Le système utilise le bit 34 du R.G. comme indicateur. Pour repassser en mode normal (TOPLEVEL) la fonction HOT.START doit donc remettre à 0 cet indicateur en évaluant (STATUS 2 34).

(HOT.START) [SUBR à 0 argument]

d'une manière standard cette fonction enlève le bit 34 du R.G. pour repasser en mode normal (TOPLEVEL) et ramène toujours la valeur NIL. HOT.START peut être redéfinie pour créer des systèmes personnels qui au moment du démarrage à chaud rentrent automatiquement dans une boucle principale non-standard.

ex :

```
? (DE HOT.START ()
?    (PRINT "Et ça repart ...")
?    (STATUS 2 34))
= HOT.START

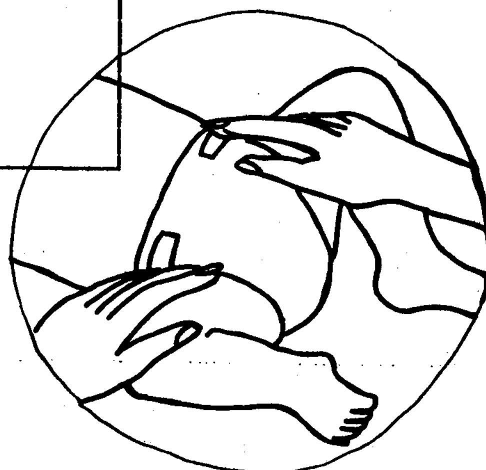
? ↑C                      au niveau TOP-LEVEL, appel du moniteur.

.NSAVE XLISP 50K         sauvetage de l'image mémoire de nom
XLISP saved                XLISP.EXE

.RUN XLISP               re-appel de l'image mémoire
  --- ALLO ? ---
  "Et ça repart ..."  exécution de la fontion HOT.START
.REE
?                    on se retrouve au niveau
                     TOP-LEVEL précédemment quitté.
```

CHAPITRE 9

ERREURS ET
MISE AU POINT



9.1 Les Erreurs

Il existe trois catégories d'erreurs :
- les erreurs détectées par le moniteur lui-même, qui sont terriblement dangereuses car elles mettent en danger l'intégrité de l'interprète.
- les erreurs détectées par l'intreprète et pour lesquelles il y a suspension autoritaire du travail après impression d'un message d'erreur (ces erreurs sont nommées erreurs fatales).
- les erreurs détectées par l'interprète mais pour lesquelles des actions spécifiques peuvent être ordonnées par l'utilisateur. Ces erreurs sont donc des erreurs non-fatales.

9.1.1 Les Erreurs Du Moniteur -

A priori il ne devrait pas y en avoir. Toutefois de mauvaises utilisations du LAP ou des fichiers provoquent encore des erreurs du type :

```
?PC OUT OF BOUNDS AT USER PC xxx
?NON-EX MEM AT USER PC xxx
?MEM PAR ERROR AT USER PC xxx

?ADDRESS CHECK FOR DEVICE y AT USER PC xxx
?IO TO UNASSIGNED CHANNEL AT USER PC xxx
```

Ces erreurs sont catastrophiques car elles retournent au moniteur. Pour re-entrer dans VLISP il faut émettre une des commandes :
- .CONT pour essayer de continuer le programme à l'endroit où il s'était arrêté. Ce n'est en général pas possible.
- .REE pour re-entrer dans l'interprète au niveau de la boucle TOP-LEVEL, en essayant de garder au maximum l'image mémoire. Dans le cas où l'interprète se trouvait dans le Gargage-collecting il devient inutilisable.
- .START pour utiliser un VLISP frais. Toutes les définitions antérieures des fonctions sont alors détruites.

9.1.2 Les Erreurs Fatales De L'interprète. -

Ces erreurs détectées par l'interprète sont sans appel. Il y a retour inconditionnel au niveau du TOPLEVEL après impression des messages. Ces messages consistent en :
- le message indiquant le type de l'erreur (parfois accompagné du ou des arguments fautifs). Ces messages commencent tous par les caractères ** .
- le libellé --- last form : <s> indiquant la dernière forme évaluée avant l'apparition de l'erreur.
- le libellé --- last fn : <s> indiquant la dernière fonction appellée (sous sa forme LAMBDA).

Ces deux dernières indications servant à repérer la fonction et l'endroit dans la fonction où l'erreur est apparue.

9.1.2.1 Erreurs Système Détectées Par L'interprète. -

** ILLEGAL UUO. Mauvaise UUO. Cette erreur apparait en général à la suite d'une mauvaise utilisation du LAP.

** ILL REF MEMORY AT USER PC : xxx mauvaise utilisation du CONS ou erreur de programmation en LAP. Cette erreur est parfois détectée après des recherches sur des objets indéterminés par exemple (CDDDR 9999).

** PDL OVERFLOW. La pile de travail de l'interprète est saturée. Cette erreur est due en général à une récursion infinie par exemple ((LAMBDA (X) (SELF X) (SELF X)) ()).

** PDL OVERFLOW DURING G.C. La pile de travail de l'interprète a été saturée au cours d'un G.C. Cette erreur oblige à recharger un VLISP frais et fait perdre toutes les définitions présentes.

9.1.2.2 Erreurs Dans Le Gestion De La Mémoire - Ces erreurs sont décrites dans le chapitre Organisation et Utilisation.

** NOT ENOUGH CORE il n'y a pas assez de mémoire pour réaliser la configuration demandée.

** NOT ROOM FOR atoms/array/code/lists/numbers/strings une des zones de stockage est saturée.

** ER G.C. STEP DONE compte du GC terminé.

** USER STACK UNDERFLOW mauvaise utilisation de la pile utilisateur.

** USER STACK OVERFLOW débordement de la pile utilisateur.

9.1.2.3 Erreurs D'entrée/sortie - Ces erreurs sont décrites dans le chapitre entrées/sorties et fichiers.

** OPEN ERROR (INPUT) : <n> erreur à l'ouverture du périphérique d'entrée.

** OPEN ERROR (OUTPUT) : <n> erreur à l'ouverture du périphérique de sortie.

** LOOKUP ERROR (INPUT) : <n> erreur à l'ouverture du fichier d'entrée.

** ENTER ERROR (OUTPUT) : <n> erreur à l'ouverture du fichier de sortie.

** LIBRARY ERROR. il y a plusieurs fichiers LIBRARY.

** READ ERROR : <n> erreur de lecture.

** READ ERROR (in LIBRARY) : <n> erreur de lecture dans un fichier LIBRARY.

** IMPLODE ERROR : <n> erreur dans la fonction IMPLODE.

9.1.2.4 Erreurs Dans Certaines Fonctions VLISP -

** ARITHMETIC EXCEPTION PC <n> erreur arithmétique.

** ARRAY ERROR <a> <n> erreur dans un tableau.

** BAD DEFINITION erreur dans les fonctions DE,DF,DM.

** ESCAPE ERROR erreur dans la fonction ESCAPE.

** LABEL ERROR <a> erreur dans la fonction GO ou GOTO.

** LESCAPE ERROR erreur dans la fonction LESCAPE.

** NON NUMERIC ARG : argument erroné dans une fonction numérique.

** RETURN ERROR erreur dans le fonction RETURN.

** SELF ERROR erreur dans la fonction SELF.

** STATUS ERROR <n> erreur dans la fonction STATUS.

** UNDEFINED FUNCTION (FSUBR) <s> erreur dans la fonction FSUBR.

** UNDEFINED FUNCTION (SUBR) <s> erreur dans la fonction SUBR.

9.1.3 Les Erreurs Non-fatales De L'interprète -

Durant l'évaluation des formes, à l'apparition de certaines erreurs, VLISP appelle des fonctions standard. La valeur de ces fonctions devient la valeur de la forme ayant provoquée l'erreur. Ces fonctions peuvent être redéfinies par l'utilisateur et permettent entre autre l'introduction de dispositifs spéciaux de traitement des erreurs, comme par exemple le dispositif connu sous le nom de PHENARETE (1). Toutes ces erreurs possèdent une grand nombre d'arguments (qui permettent toutes les fantaisies possibles au niveau de l'interprète lui-même) :
- l'atome ou la fonction ayant provoqué l'erreur,
- la valeur du pointeur de la pile de contrôle de l'interprète (son utilisation est réservée aux sorciers VLISP) sous la forme d'un nombre.
- le pointeur dans la pile des liens des blocs lambdas (son utilisation est également réservée aux wizards VLISP) sous la forme d'un nombre.
- le pointeur dans la pile des liens des blocs escapes ...

---

(1) PHENARETE étant comme chacun sait un système de correction et d'amélioration de programmes écrits en LISP. PHENARETE a été conçu et réalisé par Harald WERTZ.

9.1.3.1 Erreur Variable Indéfinie -

Cette erreur apparait si on tente d'évaluer une variable qui n'a jamais reçue de valeur.

(ERROR.UBV <at> <n1> <n2> <n3>) [SUBR à N arguments]
{pour ERROR UnBounded Variable}

<at> est l'atome qui ne possède pas de valeur et <n1> <n2> <n3> sont les pointeurs sur la pile. Cette fonction imprime d'une manière standard :
** UNDEFINED VARIABLE : <at>
dans lequel <at> est l'atome indéfini puis retourne au TOPLEVEL.

On peut redéfinir cette fonction de plusieurs manières :

1 - pour ignorer complétement ce type d'erreur et affecter systématiquement la valeur NIL à ce type de variable :

```
(DE ERROR.UBV (A)
    (SET A NIL))
```

2 - pour imprimer un message d'avertissement et affecter systématiquement la valeur NIL à ce type de variable :

```
(DE ERROR.UBV (A)
    (PRINT "Variable indéfinie" A "forcée à NIL.")
    (SET A NIL))
```

3 - pour provoquer une erreur fatale :

```
(DE ERROR.UBV (A)
    (PRINT "Variable indéfinie" A
           "dans la fonction" (LASTCALL 2))
    (RESET))
```

9.1.3.2 Erreurs Fonction Indéfinie. -

(ERROR.UDFE <s> <l> <n1> <n2> <n3>) [SUBR à N arguments]
{pour ERROR UnDefined Function in Eval}

Cette fonction est appellée automatiquement à l'apparition d'une fonction non définie dans la fonction interprète EVAL. <s> est la fonction non-définie, <l> la dernière forme évaluée et <n1> <n2> <n3> les pointeurs de pile. D'une manière standard cette fonction imprime le message :
** undefined function (EVAL) : <s>
dans lequel le nom de la fonction incriminée <s> est imprimé puis retourne au TOPLEVEL.

(ERROR.UDFA <s> <l> <n1> <n2> <n3>) [SUBR à N arguments]
{pour ERROR UnDefined Function in Apply}

Cette fonction est appellée automatiquement à l'apparition d'une fonction non-définie dans la fonction interprète APPLY. <s> est la fonction non-définie, <l> la dernière forme évaluée et <n1> <n2> <n3> les pointeurs de pile. Cette erreur n'est pas fréquente car VLISP 10 n'utilise pratiquement plus la fonction APPLY hormis dans les

fonctionnelles. D'une manière standard cette fonction imprime le message :
** undefined function (APPLY) : <s>
dans lequel le nom de la fonction incriminée <s> est imprimé puis retourne au TOPLEVEL.

Exemple de redéfinition de ces fonctions d'erreurs :

```
(DE ERROR.UDFE (fonction forme pile p$bind)
    ; Undefined function EVAL ;
    (ERROR.UDF "Fonction indéfinie dans EVAL : "))

(DE ERROR.UDFA (fonction forme pile p$bind)
    ; Undefined function APPLY ;
    (ERROR.UDF "Fonction indéfinie dans APPLY : "))

(DE ERROR.UDF (msg)
    ; Fonction générale d'erreur FUNCTION UNDEFINED ;
    (PRINT msg fonction)
    (PRINTLEVEL 6)
    (PRINTLENGTH 10)
    (PRINT "La dernière forme était : " forme)
    (OR (EQ p$bind -1)
        (PRINT "La dernière FONCTION était : " (LASTCALL 3)))
    (PRINTLEVEL 50)
    (PRINTLENGTH 2000)
    (RESET))
```

9.1.4 Les Avertissements -

L'interprète émet parfois de petits avertissements dont voici la liste succinte (leurs descriptions complètes apparaissent dans les autres chapitres).

** LEFT CELLS <n> avertissement de saturation de la mémoire liste.
** E.O.F. fin de fichier.
** E.O.F. during READ fin de fichier probablement incorrect.
** E.O.F. durind READ fin de fichier probablement incorrect dans un fichier LIBRARY.
** ESCAPE.I <n> arrivée d'une interruption de type ESCAPE I, (voir le paragraphe suivant).

9.2 Les Interruptions

Il existe dans la version IRCAM une interruption soft qui a pour nom ESCAPE.I et qui permet d'interrompre l'exécution d'un programme VLISP, de lancer une autre fonction, et de revenir au programme interrompu. Cette interruption est provoquée par l'envoi d'un code spécial par la TTY, le code ESCAPE-I qui se frappe : NUL puis I ou bien NUL puis une suite de digits décimaux puis I.

A l'apparition d'une interruption de ce type, VLISP lance une fonction de nom ESCAPE.I avec pour arguments :
1 - le nombre tapé entre NUL et I (i.e. le numéro du ESCAPE-I) s'il n'y a pas de nombre, cet argument vaut 0.
2 - le pointeur de pile lui-même
3 - le pointeur des blocs lambda dans le pile (P$BIND)

Ces arguments permettent toutes les manipulations au niveau de la pile (en particulier les restaurations de contextes) de la même manière que les erreurs non-fatales VLISP.

Cette fonction d'une facon standard imprime le libellé :
** ESCAPE-I : <n>
dans lequel <n> est le numéro du ESCAPE.I puis retourne au programme interrompu.

Il est tout-à-fait possible de redéfinir cette fonction pour gérer soi-même ce type d'interruption, de même qu'il est tout à fait possible d'appeller cette fonction au moyen de la forme (ESCAPE.I n) ce qui permet de déclencher cette interruption par programme.

Exemples d'utilisation de cette interruption :

```
; le fonctionnement minimum consiste en : ;

(DE ESCAPE.I (N)
   (INTERRUPT "Merci..."))

(DE TESTINT ()
   (ESCAPE INTERRUPT
      (PRINT "Interrompez-moi s.v.p.")
      (WHILE T ; ... en attendant l'IT ... ; )))


; On utilise donc cette interruption ;
; pour appeller des fonctions d'échappement ;

(DE ESCAPE.I () (TERMIN ... valeur à retourner ... ))

    ; et dans une fonction : ;
    (ESCAPE TERMIN
       (WHILE T
          ...
       ))
    ; endroit où l'on veut recevoir une valeur et revenir ;
    ; à l'occurence d'une interruption ESCAPE.I ;


; Il est également possible de fabriquer un BREAK dynamique ;

(DE ESCAPE.I (numero pile p$bind)
        (TERPRI)
        (PRINT "Je rentre dans un TOPLEVEL ESCAPE-I.")
        (PRINT "Pour en sortir, commence une ligne par <META-ESPACE>.")
        (STATUS 11 '/!)
        (TEREAD)
        (UNTIL (EQ (PEEKCH) '/ )
           (PRINT (EVAL (READ))))
        (STATUS 11 '/?)
        (PRINT "Ca roule..."))
```

9.3 Les Traces

Une des techniques les plus usuelles de mise au point de programmes VLISP est le "traçage" des fonctions. "Tracer" une fonction consiste à faire imprimer, à l'entrée de celle-ci et après évaluation de ses arguments, le symbole ---> suivi de son nom ainsi que la valeur de ses arguments, et au retour de la fonction à faire imprimer le symbole <--- suivi de son nom ainsi que la valeur retournée. Ces traces permettent de visualiser les appels dynamiques des fonctions.

9.3.1 Trace Des Fonctions De L'interprète -

Il est possible de tracer certaines fonctions internes de l'interprète en positionnant des bits du R.G. :

bit 3 du R.G.
si ce bit est positionné (ce qui n'est pas le cas par défaut), trace tous les appels internes de la fonction EVAL. Cette fonction qui est utilisée pour chaque évaluation produit des traces fort longues et est en général utilisée pour mettre au point l'interprète lui-même ou pour fonctionner en mode pas-à-pas.

bit 4 du R.G.
si ce bit est positionné (ce qui n'est pas le cas par défaut), trace tous les appels internes de la fonction APPLY. Cette fonction est très peu employée par l'interprète car, en VLISP, c'est EVAL qui traite les LAMBDA-expressions.

ex : si le bit 3 est positionné, l'évaluation de :

```
(PROGN
   (SETQ X 4)
   (SETQ Y (ADD1 X)))
```

produira la trace :

```
---> EVAL : (PROGN (SETQ X 4) (SETQ Y (ADD1 X)))
---> EVAL : (SETQ X 4)
---> EVAL : 4
<--- EVAL : 4
<--- EVAL : 4
---> EVAL : (SETQ Y (ADD1 X))
---> EVAL : (ADD1 X)
---> EVAL : X
<--- EVAL : 4
<--- EVAL : 5
<--- EVAL : 5
<--- EVAL : 5
```



9.3.2 Trace Automatique Des Fonctions Standard -

Il est également possible de tracer automatiquement n'importe quelle fonction standard (de type SUBR ou FSUBR) au moyen d'appels à des fonctions STATUS.

(STATUS 28 <at>) [SUBR à N arguments]

trace la fonction standard de nom <at>. Si <at> n'est pas un atome littéral, l'erreur STATUS se déclenche.

(STATUS 29 <at>) [SUBR à N arguments]

enlève la trace de la fonction standard de nom <at>. Si <at> n'est pas un atome littéral, l'erreur STATUS se déclenche.

Ces deux fonctions sont en général utilisées pour tester les fonctions standard elles-mêmes.

9.3.3 Trace Manuelle Des Fonctions Standard -

Il est possible de redéfinir sous forme de EXPR ou de FEXPR les fonctions standard et d'avoir simultanément accès à une version interprétée et à une version codée d'une même fonction standard. L'appel de la version interprétée se fera normalement au moyen du nom de la fonction tandis que l'appel de la version codée se fera au moyen des fonctions SUBR ou FSUBR (voir ces fonctions).
Les traces de ce type sont beaucoup plus souples que les traces automatiques et permettent de filtrer commodément les configurations intéressantes.

ex : tracer tous les appels de la fonction ASSQ
qui s'effectuent à l'intérieur de la fonction FOO
et qui utilisent une A-liste de nom ENV

```
(DE ASSQ (a al)
   (IF (AND (EQP (LASTCALL 2) (GET 'FOO EXPR))
            (EQ al 'ENV))
       (PRINT "Recherche sur ENV dans FOO"
              a al '= (SUBR 'ASSQ [a al]))
       (SUBR 'ASSQ [a al])))
☞ ASSQ

(SETQ ENV '((A . 1) (B . 2)) ALST '((X . 9)))
☞ ((X . 9))

(ASSQ 'X ALST)  ☞  9

(DE FOO (X) (ASSQ X 'ENV))  ☞  FOO

(FOO 'A)
"Recherche sur ENV dans FOO" A ((A . 1)(B . 2)) = 1
☞ 1
```

9.3.4 Trace Généralisée -

Les fonctions qui vont être décrite permettent de tracer n'importe quel type de fonction (SUBR FSUBR EXPR FEXPR MACRO MACIN MACOUT ARRAY). Elles sont de type AUTOLOAD (i.e. il n'est pas besoin de charger le fichier qui les contient, le système le fera pour vous automatiquement au premier appel de l'une de ces fonctions).
D'ordinaire ces fonctions se trouvent dans le fichier :
    (SYS (DEBUG . VLI)) dans leurs versions interprétées
Ce fichier est donné en Appendice C.

(TRACE <at1> ... <atN>) [FEXPR]

indique qu'il faut tracer les différentes fonctions de nom <at1> ... <atN>. Si L'un de ces atomes n'est pas une fonction de type SUBR, FSUBR, EXPR, FEXPR, MACRO, MACIN, MACOUT ou ARRAY le message d'avertissement suivant apparait :
    ** TRACE : c'est quoi <at>
dans lequel <at> est le nom de l'atome incriminé. Si la fonction est de type EXPR, FEXPR, MACRO, MACIN ou MACOUT alors un nombre est imprimé à gauche des flèches <--- et ---> ; il indique la profondeur d'appel récursive de la fonction tracée. En plus du traçage, cette fonction permet donc d'étudier les appels dynamiques d'une fonction donnée. TRACE ramène en valeur la liste des fonctions actuellement tracées.

(UNTRACE <at1> ... <atN>) [FEXPR]

enlève la trace des différentes fonctions de nom <at1> ... <atN>. Si aucun nom n'est fourni (i.e. si on évalue (UNTRACE)), UNTRACE enlève la trace de toutes les fonctions précédement tracées. Si un atome n'est pas une fonction de type SUBR, FSUBR, EXPR, FEXPR, MACRO, MACIN, MACOUT ou ARRAY le message d'avertissement apparait :
    ** UNTRACE : c'est quoi <at>
dans lequel <at> est le nom de l'atome incriminé. UNTRACE ramène en valeur la liste des fonctions dont on vient d'enlever la trace.

(TRACEFILE <filin>) [EXPR à 1 argument]

lit le fichier <filin> et prépare le traçage de toutes les définitions (de type DE DF DM DMI DMO) qui s'y trouvent. TRACEFILE est utilisée pour analyser les fonctions de tout un fichier et ramène <filin> en valeur.

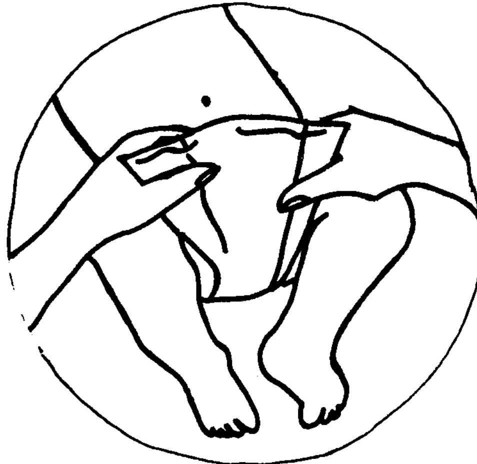
(TRACEF <filin>) [FEXPR]

est une forme abrégée de la fonction TRACEFILE. L'appel de (TRACEF <file>) correspond à l'appel (TRACEFILE '<file>).

9.3.5 Exemple D'utilisation Des Traces -

```
? ; la sacro-sainte fonction FOO ;
?
? (DE FOO (N)
?    (IF (ZEROP N) 1 (TIMES N (FOO (SUB1 N)))))
= FOO
= ; time = 0 ms ;
? (TRACE FOO GLUCK TIMES)
```

```
DEBUG loaded.
   "** TRACE : C'est quoi " GLUCK
=  (FOO TIMES)
=  ; time = 480 ms ;
?  (FOO 3)
   1 ---> FOO    (3)
   2 ---> FOO    (2)
   3 ---> FOO    (1)
   4 ---> FOO    (0)
   4 <--- FOO    1
   ---> TIMES :  (1 1)
   <--- TIMES =  1
   3 <--- FOO    1
   ---> TIMES :  (2 1)
   <--- TIMES =  2
   2 <--- FOO    2
   ---> TIMES :  (3 2)
   <--- TIMES =  6
   1 <--- FOO    6
=  6
=  ; time = 140 ms ;
?  (UNTRACE)
=  (FOO TIMES)
=  ; time = 0 ms ;
?  (FOO 3)
?  (FOO 3)
=  6
=  ; time = 20 ms ;
?  ←
Bye
EXIT
.
```



9.4 Le Pas à Pas

Exécuter un programme en pas à pas consiste à imprimer chaque forme avant son évaluation par la fonction EVAL (d'une manière identique à celle obtenue après la mise du bit 3 du R.G.), à imprimer ensuite le signal // sur le terminal puis à se mettre en lecture d'une commande qui sera exécutée avant évaluation de la forme.
Ces commandes servent principalement à sélecter les traces désirées, ce qui supprime l'inconvénient majeur de la trace classique : la taille considérable des impressions.
Cette exécution en pas à pas est obtenue en positionnant simultanément le bit 3 (de trace) et le bit 8 du R.G. .

9.4.1 Les Commandes Du Pas à Pas -

toutes ces commandes sont codées avec une seule lettre et doivent être tapées après le code // .

//> continue le traçage récursif de tous les objets évalués.

//< arrête momentanément le traçage, évalue la dernière expression imprimée, imprime sa valeur et reprend le traçage normalement.

//. arrête définitivement le traçage.

//P reimprime la dernière expression

//= passe en mode EVAL , i.e. lit une S-expression, l'évalue, imprime son résultat puis repasse en mode pas à pas.

9.4.2 Les Fonctions Du Pas à Pas -

Toutes les fonctions qui vont être décrites sont de type AUTOLOAD i.e. il n'est pas besoin de charger le fichier qui les contient, le système le fera pour vous automatiquement au premier appel de l'une de ces fonctions.
D'ordinaire ces fonctions se trouvent dans le fichier :
    (SYS (DEBUG . VLI))
Ce fichier est donné en Appendice C.

(STEPALL <i>) [FEXPR]

si l'indicateur <i> est positionné (si <i> = T), l'interprète passe en mode pas à pas, si cet indicateur n'est pas positionné (si <i> = NIL) l'interprète repasse en mode normal.

(STEP <at1> ... <atN>) [FEXPR]

indique de passer en mode pas à pas à l'entrée de chacune des fonctions <at1> ... <atN>. Si l'un de ces atomes littéraux n'est pas une fonction de type EXPR ou FEXPR, le message d'avertissement apparait :
    ** STEP : C'est quoi <at>
dans lequel <at> est le nom de l'atome incriminé. STEP ramène en valeur la liste des fonctions qu'il faut exécuter en pas à pas.

(UNSTEP <at1> ... <atN>) [FEXPR]

enlève le passage en mode pas à pas à l'entrée des fonctions <at1> ... <atN>. Si aucun nom n'est fourni (i.e. si on évalue (UNSTEP)), UNSTEP enlève tous les passages en pas à pas existants. Si l'un de ces atomes litéraux n'est pas une fonction de type EXPR ou FEXPR, le message d'avertissement apparait :
    ** UNSTEP : C'est quoi <at>
dans lequel <at> est le nom de l'atome incriminé. UNSTEP ramène en valeur la liste des fonctions qu'il ne faut plus exécuter en pas à pas.

9.4.3 Exemples De Pas à Pas -

```
? ; définissons une fonction de test ;
?
?  (DE EK (X Y)
?     (COND
?        ((ATOM X) (EQ X Y))
?        ((ATOM Y) NIL)
?        (T (AND (EK (CAR X) (CAR Y))
?                (EK (CDR X) (CDR Y))))))
=  EK
=  ; time = 0 ms ;
?
```

```
? ; demande de STEP inconditionnel ;
? (STEPALL T)
SYS:DEBUG.VLI loaded.
= T
= ; time = 620 ms ;
?
? ; le STEP est maintenant actif ;
? ; 1er exemple de STEP ;
? ()
  ---> EVAL  : NIL  // >
  <--- EVAL  = NIL
= NIL
= ; time = 20 ms ;
?
? ; appel sous STEP de la fonction précédemment définie ;
?
? (EK '(A (B C) D) '(A (B C) D))
  ---> EVAL  : (EK '(A (B C) D) '(A (B C) D))  // >
  ---> EVAL  : '(A (B C) D)  // >
  <--- EVAL = (A (B C) D)
  ---> EVAL  : '(A (B C) D)  // >
  <--- EVAL = (A (B C) D)
  ---> EVAL  : (COND ((ATOM X) (EQ X Y)) ((ATOM Y) NIL) (T (AND & &)))  // P
   (COND ((ATOM X) (EQ X Y)) ((ATOM Y) NIL) (T (AND (EK (CAR X) (CAR
  Y)) (EK (CDR X) (CDR Y)))))  // >
  ---> EVAL  : (ATOM X)  // <
  <--- EVAL = NIL
  ---> EVAL  : (ATOM Y)  // <
  <--- EVAL = NIL
  ---> EVAL  : T  // <
  <--- EVAL = T
  ---> EVAL  : (AND (EK (CAR X) (CAR Y)) (EK (CDR X) (CDR Y)))  // >
  ---> EVAL  : (EK (CAR X) (CAR Y))  // =
? X
   (A (B C) D)  // =
? Y
   (A (B C) D)  // P
   (EK (CAR X) (CAR Y))  // <
  <--- EVAL = T
  ---> EVAL  : (EK (CDR X) (CDR Y))  // >
  ---> EVAL  : (CDR X)  // >
  ---> EVAL  : X  // >
  <--- EVAL = (A (B C) D)
  <--- EVAL = ((B C) D)
  ---> EVAL  : (CDR Y)  // >
  ---> EVAL  : Y  // >
  <--- EVAL = (A (B C) D)
  <--- EVAL = ((B C) D)
  ---> EVAL  : (COND ((ATOM X) (EQ X Y)) ((ATOM Y) NIL) (T (AND & &)))  // p
   (COND ((ATOM X) (EQ X Y)) ((ATOM Y) NIL) (T (AND (EK (CAR X) (CAR
  Y)) (EK (CDR X) (CDR Y)))))  // .
  <--- EVAL = T
  <--- EVAL = T
  <--- EVAL = T
  <--- EVAL = T
  <--- EVAL = T
= T
= ; time = 720 ms ;
?
? ; le STEP est maintenant inactif ;
? ()
= NIL
= ; time = 0 ms ;
```

?

9.5 Les Comptages

Il est souvent utile de connaître combien de fois une fonction a été appellée en cours d'exécution d'un programme VLISP. En effet c'est aux fonctions appellées le plus souvent qu'il faut porter le plus grand intérêt au moment de l'amélioration (voire de l'optimisation) des programmes. Il existe des fonctions standard qui vont permettre de compter automatiquement le nombre d'appels dynamiques de vos fonctions.
Toutes les fonctions qui vont être décrites sont de type AUTOLOAD i.e. il n'est pas besoin de charger le fichier qui les contient, le système le fera pour vous au premier appel de l'une de ces fonctions.
D'ordinaire ces fonctions se trouvent dans le fichier :
    (SYS (DEBUG . VLI))
Ce fichier est donné en Appendice C.

(COUNT <at1> ... <atN>) [FEXPR]

indique qu'il faut effectuer un comptage automatique du nombre d'appels de chacune des fonctions
<at1> ... <atN>. Si l'un de ces atomes litéraux ne possède pas de définition de type EXPR ou FEXPR, le message d'avertissement suivant apparait :
    ** COUNT : C'est quoi <at>
dans lequel le nom de l'atome incriminé est imprimé. Cette fonction ramène la liste (<at1> ... <atN>) en valeur.

(UNCOUNT <at1> ... <atN>) [FEXPR]

provoque l'arrêt du comptage automatique des appels des fonctions
<at1> ... <atN> et imprime le résultat de ce comptage sous la forme :
    <at1> : n
    ... : ..
    <atN> : n
dans laquelle les atomes sont triés par ordre alphabétiques.
UNCOUNT ramène la liste (<at1> ... <atN>) en valeur.

(COUNTFILE <filin>) [EXPR à 1 argument]

lit le fichier <filin> et prépare le comptage de toutes les définitions (de type DE DF DM DMI DMO) qui s'y trouvent. COUNTFILE est utilisé pour analyser les fonctions de tout un fichier et ramène <filin> en valeur.

(COUNTF <filin>) [FEXPR]

est une forme abrégée de la fonction COUNTFILE. L'appel de (COUNTF <file>) correspond à l'appel (COUNTFILE '<file>).

```
; Exemple d'utilisation des comptages pour les fonctions
  %EOL %PRIN1 FLATSIZE et SUPERP du PRETTY-PRINT
  qui se prettyprinte lui-même! ;

?  ; on suppose que le Pretty-print est déjà chargé en mémoire ;
?
```

```
?  (COUNT %EOL %PRIN1 FLATSIZE SUPERP)   ; demande du comptage ;
SYS:DEBUG.VLI loaded.
=  (%EOL %PRIN1 FLATSIZE SUPERP)
=  ; time = 1000 ms ;
?  (PRETTYF PRETTY)        ; appel du pretty-print sur lui-même ;

  --- ALLO ? ---
=  (DSK (PRETTY . VLP) (LIS . JER) 45)
=  ; time = 34820 ms ;
?  (UNCOUNT)               ; demande les résultats du comptage ;

   %EOL :          472
   %PRIN1 :        1580
   FLATSIZE :      5188
   SUPERP :        1505

=  (%PRIN1 FLATSIZE %EOL SUPERP)
=  ; time = 240 ms ;
?
?


; Nouvel exemple de COUNT : comptage de toutes les fonctions
  du compilateur durant la compilation du fichier DISPLAY ;


?
? (COUNTF COMPIL)
SYS:DEBUG.VLI loaded.
SYS:COMPIL.VLI loaded.

  --- ALLO ? ---
=  COMPIL
=  ; time = 17180 ms ;
?
?  (COMPILEF DISPLAY T T)
SYS:PRETTY.VLI loaded.

  --- ALLO ? ---
=  (DSK (DISPLAY . VLA) (LIS . JER) 45)
=  ; time = 12140 ms ;
?
?  (UNCOUNT)
   *18BITP :       6
   *ADD :          37
   *ADD1 :         282
   *CMP :          160
   *CMPRED :       5
   *CMPREDSTD :    1
   *GEN :          97
   *GEN0SUBR :     2
   *GEN1SUBR :     19
   *GEN2SUBR :     20
   *ISKIP :        9
   *KST :          35
   *LENL :         8
   *LENS :         178
   *LKT :          2
   *OPT :          46
   *PA1 :          13
   *PROGN :        10
   *RESOL :        8
   *RSTL :         1
```

```
   *SSB :        8
   *SSB1 :       2
   *TWO :        24
   *TYPFNT :     8
   COMPIL :      8
   COMPILEF :    1
   COMPILEFILE : 1
   COMPILEREASE : 8
   COMPILES :    8
   COMPILOPTIONS 1
   COMPILPR :    8
   DGEN :        1
   MACMP :       14
   MEXPAND2 :    7
   MEXPARITH :   14
   PRCOD :       2
   PRCOMMENT :   14
   PRLAP :       8
   PRSSEC :      8
   PRTBL :       8
=  (*18BITP *ADD *ADD1 *CMP *CMPRED *CMPREDSTD *DGEN2 *GEN *GEN0SUBR
= *GEN1SUBR *GEN2SUBR *ISARG *ISKIP *KST *LAMBDA *LENL *LENS *LK0 *LK1 *LKM
= *LKSTBL *LKT *NIEWIEM *OPT *PA1 *PKST *PROGN *RCR *RESOL *RSTL *SCR *SSB
= *SSB1 *SSBP *TWO *TYPFNT *UNIQUEP COMPIL COMPILE COMPILEF COMPILEFILE
= COMPILEND COMPILEREASE COMPILERROR COMPILES COMPILINDIC COMPILOPTIONS
= COMPILPR DGEN MACMP MEXPAND MEXPAND2 MEXPARITH PRCOD PRCOMMENT PRLAP PRSSEC
= PRTBL SORTL TMACMP)
=  ; time = 1860 ms ;
?  ←
Bye
EXIT
.
```

9.6 Autres Fonctions De Mise Au Point

(LOC <s> <i>) [SUBR à 2 arguments]

permet de connaître l'adresse mémoire de l'objet VLISP <s>. Si l'indicateur <i> = T, cette adresse est l'adresse physique réelle, si cet indicateur = NIL, c'est l'index par rapport au début des objets VLISP qui est retourné en valeur.

```
ex : (LOC NIL)     ☞  0
     (LOC 'ZOO)    ☞  2760
     (LOC NIL T)   ☞  1246
```

(VAG <n> <i>) [SUBR à 2 arguments] {pour VAlue Get}

ramène l'objet VLISP d'adresse <n>. Si l'indicateur <i> = T, cette adresse est l'adresse physique réelle, sinon <n> est un index par rapport au début des objets VLISP. Cette fonction est utilisée conjointement à la fonction LOC pour manipuler les adresses physiques.

```
ex : (VAG 0)                ☞  NIL
     (VAG 2760)             ☞  ZOO
     (VAG (LOC '(G O N)))   ☞  (G O N)
```

(PATCH <n1> <n2>) [SUBR à 2 arguments]

permet de modifier le contenu du mot mémoire d'adresse <n1> avec la valeur <n2> (si <n2> est donné). PATCH ramène le contenu du mot mémoire d'adresse <n1> après modification. Il est à noter que cette fonction utilise l'UUO SETUWP ce qui permet de modifier le segment haut de l'interprète lui-même. Cette pratique, très pernicieuse, est à éviter dans la mesure du possible.

(MEMORY <n>) [SUBR à 1 argument]

ramène la valeur du mot mémoire d'adresse <n>. Il est préférable d'utiliser cette fonction à la place de la fonction précédente (PATCH), si la mémoire est en lecture seule, car cette fonction ne peut pas provoquer d'accidents pour l'interprète lui-même.

(DDT) [SUBR à 0 argument]
{pour Dichloro Difluoro Toluène}

Appel l'utilitaire DDT si celui-ci a été chargé. Cet utilitaire sert à mettre au point les programmes écrits en langage machine (interprète ou fonctions LAP). Pour revenir sous VLISP il faut utiliser la commande RDDT$X sous DDT. Si DDT n'est pas chargé cette fonction ramène NIL tout de suite.



CHAPITRE 10

PRETTY PRINT
CROSS REFERENCE
ET INDEX

10.1 Le Pretty-Print

Les fonctions standards PRINT et PRIN1 sont d'ordinaire utilisées pour éditer les S-expressions VLISP. Les seules mesures prises pour améliorer la lisibilité sont :
- l'insertion d'un espace entre chaque atome ;
- l'interdiction d'éditer un atome (atome littéral, nombre ou chaîne de caractères) sur deux lignes.

Ces mesures sont nettement insuffisantes pour éditer vos programmes. Les fonctions du Pretty-Print vont les éditer d'une manière beaucoup plus lisible en faisant resortir, au moyen de renfoncements gauches et de sauts de lignes ad hoc, la structure de vos fonctions.
Les macros-caractères définis de manière standard dans le système (i.e. les caractères ' [ ]) sont restitués par le Pretty-Print; en particulier :

```
(QUOTE <s>)             devient   '<s>
(NCONS <s>)             devient   [<s>]
(CONS <s1> <s2>)        devient   [ <s1> . <s2> ]
(MCONS <s1> ... <sN>)   devient   [ <s1> ... <sN-1> . <sN> ]
(LIST <s1> ... <sN>)    devient   [ <s1> ... <sN> ]
```

En outre les LAMBDA-expressions explicites sont éditées sous leurs formes LET.

10.1.1 Les Fonctions Du Pretty-Print -

Les fonctions standard qui vont être décrites sont de type AUTOLOAD (i.e. il n'est pas besoin de charger le fichier qui les contient, le système le fera pour vous au premier appel de l'une de ces fonctions).
D'ordinaire ces fonctions se trouvent dans le fichier :

```
(SYS (PRETTY . VLI)) pour la version interprétée
(SYS (PRETTY . VLA)) pour la version compilée lisible
(SYS (PRETTY . VLO)) pour la version compilée chargeable
```

Ce fichier est donné en Appendice D.

Il existe une image mémoire contenant le Pretty-Print compilé. Cette image mémoire se trouve (en général) dans un fichier système de nom : VPRETTY . Donc pour utiliser un interprète contenant déjà un PRETTY-PRINT compilé il suffit d'émettre la commande moniteur :

```
.R VPRETTY
```

(PRETTYP <s>) [SUBR à 1 argument]

édite la S-expression <s>. Cette fonction ramène <s> en valeur. Elle est utilisée en général dans vos propres fonctions qui veulent éditer proprement des S-expressions VLISP.

(PRETTY <a1> <a2> ... <aN>) [FSUBR]

édite les définitions existantes de type EXPR, FEXPR, MACRO, MACIN, MACOUT ou macro-caractère des différents atomes <a1> ... <aN> sous leurs formes DE, DF, DM, DMI, DMO ou DMC. Cette fonction ramène toujours NIL en valeur. Elle est utilisée en mise au point conversationnelle, pour vérifier aisément un parenthèsage douteux.

(PRETTYFILE <filout> <filin> <sw1>) [SUBR à 3 arguments]

édite dans le fichier de sortie <filout>, toutes les S-expressions contenues dans le fichier d'entrée <filin>. S'il n'y a pas de spécification pour le fichier de sortie (si <filout> = NIL), le fichier (DSK (PRETTY . VLP) () \055) est utilisé. L'extension standard des fichiers de sortie du Pretty-Print est .VLP . Ces fichiers peuvent être relus par VLISP, mais doivent être reformatés pour utiliser l'éditeur de l'IRCAM ETV.
L'indicateur <sw1> permet de contrôler le transcodage automatique minuscule majuscule : si <sw1> = NIL, toutes les lettres sont traduites en majuscule, si <sw1> = T les lettres sont restituées dans les casses d'origine. Si cette dernière option est choisie, les noms des fonctions standard ne sont pas reconnus s'ils apparaissent en minuscules.
Cette fonction permet de faire des copies lisibles de vos programmes après mise au point. De plus cette fonction a le bon goût de lire et de reécrire les commentaires (i.e. la suite de caractères quelconques encadrée du délimiteur ";") qui sont d'habitude complétement ignorés de VLISP. Ceci devrait vous donner l'envie de bien commenter les programmes.
Un saut de page est effectué à chaque occurence du caractère ↑L (Form Feed) ou à la rencontre du commentaire spécial ;PAGE; .
PRETTYFILE ramène en valeur le nom du fichier de sortie utilisé.

(PRETTYF <file> <sw1>) [FSUBR]

est une forme abrégée de la fonction PRETTYFILE. L'appel (PRETTYF <file> <sw1>) correspond à l'appel suivant :

```
   (PRETTYFILE '(DSK (<file> . VLP) (GETPPN) \055)
          '(DSK (<file> . VLI))
          <sw1>)
```

Cette fonction est donc utilisée pour éditer d'une manière standard un fichier standard. PRETTYF ramène la valeur du PRETTYFILE i.e. le nom du fichier de sortie utilisé.

(PRETTYSIZE <n>) [SUBR à 1 argument]

permet d'initialiser à <n> la largeur d'impression du Pretty-Print. Par défaut cette largeur est de 72 ce qui correspond à-peu-près au format standard 21x29,7. Si <n> n'est pas donné ou n'est pas un nombre, la largeur standard est prise. PRETTYSIZE ramène la largeur d'impression courante du Pretty-Print.

(PRETTYEND) [SUBR à 0 argument]

permet de récuperer la place occupée par les fonctions du PRETTY-PRINT. PRETTYEND remet également les indicateurs AUTOLOAD de ces fonctions. PRETTYEND ramène en valeur le nombre de doublets libérés. Si vous utilisez le Pretty-Print compilé, cette fonction n'a aucun effet.

10.1.2 Edition Des Fonctions De L'utilisateur -

Le Pretty-Print n'a pas à priori de connaissances sur les fonctions de l'utilisateur; celles-ci sont éditées sans formatage spécial. Il est possible d'indiquer au Pretty-Print le format souhaité pour des fonctions particulières de l'utilisateur en mettant sur la P-liste des atomes fonctions le format à utiliser sous l'indicateur PRETTY. Le type du format à utiliser est simplement le nom de la fonction standard dont on veut "imiter" le format. Il n'est pas possible actuellement de définir de nouveaux formats.

ex : (PUT 'US3 'PROGN 'PRETTY)

indique au Pretty-Print d'éditer la fonction US3 d'une manière identique à la fonction standard PROGN.

Les formats les plus usités sont :



```
PROGN      (<fonction>
               <argument 1>
                 .....
                 .....
               <argument N>)

LAMBDA     (<fonction> <argument 1>
               <argument 2>
                 .....
                 .....
               <argument N>)

DE         (<fonction> <argument 1> <argument 2>
               <argument 3>
                 .....
                 .....
               <argument N>)

SETQ       (<fonction>
               <argument 1>  <argument 2>
                 .................
                 .................
               <argument N-1> <argument N>)
```

Pour ne pas surcharger la P-liste des atomes par des indicateurs qui sont rarement utilisés, il est recommandé de ne mettre ces indicateurs qu'au moment de l'édition. La fonction PRETTYFILE (ainsi que la fonction PRETTYF) interprète les formes :

(POUR PRETTY ...)

La pose des indicateurs spéciaux PRETTY se fera donc avantageusement par cet intermédiaire.

ex : (POUR PRETTY (PUT 'US3 'PROGN 'PRETTY))

Cette forme, si elle est lue par la fonction PRETTYFILE, est équivalente à (PUT 'US3 'PROGN 'PRETTY) en revanche, lue par le TOPLEVEL (i.e. l'évaluateur VLISP), elle est ineffective.

Cette interprétation des POUR peut avoir d'autres utilités que la pose d'indicateurs
par exemple :

(POUR PRETTY (STATUS 6 8))
indique au Pretty-Print (et à lui seul) qu'il faut écrire les nombres en utilisant une base octale ...

10.1.3 Exemples D'Utilisation Du Pretty-Print -

```
.R VLISP
   VLISP 10.3-21  5-Oct-78 23:44:23 (LIS . JER)
   Pour avoir l'etat de la documentation demande .HELP VLISP
SYS:VLISP.INI loaded.
DSK:VLISP.INI loaded.
  ---  ALLO ?  ---
?
?
?  (PRETTY WHOIS WHOISALL)
SYS:PRETTY.VLI loaded.

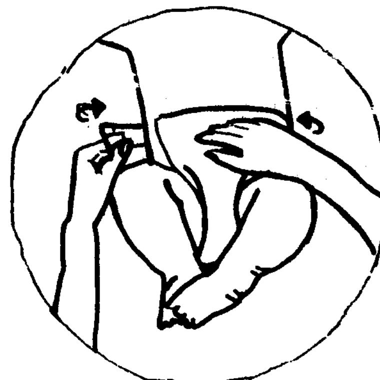
   (DF WHOIS (NAME LIGNE JELAI)
       (SETQ NAME (CAR NAME))
       (WITH (EOF NIL (INPUT) (&EOF))
          (FILE READ "SYS:FACT.TXT" T)
          (ESCLOOP &EOF
             (SETQ LIGNE (READSTR))
             (ESCAPE &UNCOUP
                (MAPC (IMPLODE (CONCAT "(" LIGNE ")"))
                   (LAMBDA (NOM)
                      (IF (SAMEPN NOM NAME)
                         (&UNCOUP (PRINT LIGNE) (SETQ JELAI T))))))))
       (IF JELAI 'OK "Nie ma ..."))


   (DE WHOISALL ()
       (WITH (EOF NIL (INPUT) (&EOF))
          (FILE READ "SYS:FACT.TXT" T)
          (ESCLOOP &EOF (PRINT (READSTR))))
       'OK)

 =  NIL
 =  ; time = 5660 ms ;
?
?
?  (PRETTY /ↄ)

   (DMC /ↄ () ['PRETTY (READ)])

 =  NIL
 =  ; time = 80 ms ;
?
?  ←
Bye
EXIT
.
```



10.2 Le Cross-Référence

D'une manière identique au Pretty-Print, le Cross-Référence va éditer des fonctions VLISP, en faisant ressortir au moyen de renfoncements gauches et de sauts de lignes ad hoc, les structures de contrôle des fonctions.
De plus chaque ligne en sortie est numérotée, ce qui permet en fin d'édition d'établir un listage des références croisées, i.e. la liste de tous les atomes litéraux, triée par ordre alphabétique, rencontrés dans la (ou les) fonction d'entrée. A côté de chaque atome, les numéros de lignes, dans lesquelles apparait cet atome, sont édités, triés par ordre croissant.

10.2.1 Les Fonctions Du Cross-Référence -

Les fonctions du Cross-Référence qui vont être décrites sont de type AUTOLOAD (i.e. il n'est pas besoin de charger le fichier qui les contient, le système le fera pour vous au premier appel de l'une de ces fonctions).
D'ordinaire ces fonctions se trouvent dans le fichier :

```
   (SYS (PRETTY . VLI)) pour la version interprètée
   (SYS (PRETTY . VLA)) pour la version compilée lisible
   (SYS (PRETTY . VLO)) pour la version compilée chargeable
```

Ce fichier est donné en Appendice D.

Les fonctions du Cross-Référence se trouvent donc sur le même fichier que les fonctions du Pretty-Print. Il est donc possible d'utiliser l'image mémoire contenant ce fichier compilé. L'appel de cette image mémoire s'effectue au moyen de la commande moniteur :

```
   .R VPRETTY
```

(CROSSFILE <filout> <filin> <sw1> <sw2>) [SUBR à N arguments]

> édite dans le fichier de sortie <filout>, toutes les S-expressions contenues dans le fichier d'entrée <filin> en numérotant toutes les lignes, puis toujours dans le même fichier de sortie CROSSFILE imprime la liste des références croisées de tous les atomes litéraux du fichier.
> S'il n'y a pas de spécification pour le fichier de sortie (si <filout> = NIL), le fichier (DSK (CROSS . VLC) () \055) est utilisé.
> L'extension standard des fichiers de sortie du Cross-Référence est .VLC .
> Ces fichiers ne peuvent pas être relus directement par VLISP car les lignes sont numérotées.
> L'indicateur <sw1> sert a spécifier s'il faut traiter les références aux atomes systèmes. Si <sw1> = NIL, seuls les atomes utilisateurs seront traités, si <sw1> = T, tous les atomes seront traités.
> L'indicateur <sw2> permet de contrôler le transcodage automatique minuscule majuscule : si <sw2> = NIL, toutes les lettres sont traduites en majuscules, si <sw2> = T les lettres sont restituées dans leurs casses d'origine. Si cette dernière option est choisie, les noms des fonctions standard ne sont pas reconnus s'ils apparaissent en minuscule.
> CROSSFILE ramène le nom du fichier de sortie utilisé.

(CROSSF <file> <sw1> <sw2>) [FSUBR]

> est une forme abrégée de la fonction CROSSFILE. L'appel (CROSSF <file> <sw1>) correspond à l'appel suivant :

```
   (CROSSFILE '(DSK (<file> . VLC) (GETPPN) \055)
              '(DSK (<file> . VLI))
              <sw1>
              <sw2>)
```

Cette fonction est donc utilisée pour éditer d'une manière standard un fichier standard. CROSSF ramène la valeur du CROSSFILE i.e. le nom du fichier de sortie utilisé.

(CROSS <at> <sw1> <sw2>) [FSUBR]

Effectue l'édition de la fonction de nom <at> ainsi que l'impression de ses références croisées d'une manière identique à la fonction CROSSFILE. Toutes ces impressions se font dans le fichier de sortie courant. Les indicateurs <sw1> et <sw2> sont utilisés de manière identique.

10.2.2 Exemple D'utilisation Du Cross-Référence -

```
.R VPRETTY

 ---  ALLO ?  ---
.REE
?
?  (CROSS TYPE T)

  1      (DE TYPE (FILIN)
  2         (FILE READ FILIN)
  3         (STATUS 17 (ASCII 13) 2)
  4         (WITH (EOF NIL (TERPRI) (INPUT) (&EOF))
  5            (ESCLOOP &EOF (PRINC (READCH))))
  6         (STATUS 17 (ASCII 13) 0)
  7         FILIN)


                    CROSS REFERENCE


    1   &EOF          4 5

    2   ASCII         3 6

    3   DE            1

    4   EOF           4

    5   ESCLOOP       5

    6   FILE          2

    7   FILIN         1 2 7

    8   INPUT         4

    9   PRINC         5

    10  READ          2

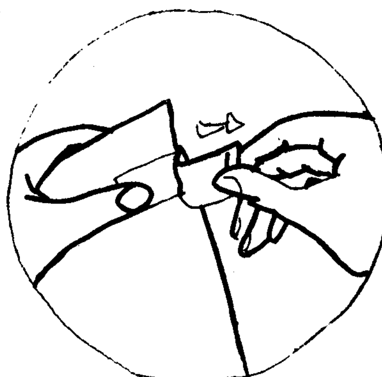
    11  READCH        5

    12  STATUS        3 6

    13  TERPRI        4

    14  TYPE          1

    15  WITH          4
```



10.3 L'Index (1)

Le programme d'INDEX effectue la lecture d'un ensemble de fonctions et édite, en fin de lecture, une série de renseignements sur chacunes des fonctions traitées ainsi que sur certaines interactions entre les fonctions elles-même.

L'INDEX fournit pour chaque fonction les renseignements suivants :

TYPE = le type de la fonction qui peut être EXPR, FEXPR, MACRO, MACIN, MACOUT ou MCHAR (pour les macro-caractères).

ARGS = les arguments de la fonction

FVARS = les variables libres utilisées à l'intérieur de cette fonction.

FVARSET = les variables libres qui sont modifiées à l'intérieur de cette fonction.

STRINGS = les constantes de chaîne de caractères utilisées dans cette fonction.

USING = les fonctions (de l'ensemble des fonctions traitées uniquement) qui sont appellées par cette fonction.

USEDBY = les fonctions (de l'ensemble des fonctions traitées uniquement) qui appellent cette fonction.

Ces deux derniers renseignements sont indispensables pour déterminer l'arbre des appels des fonctions.

10.3.1 Les Fonctions De L'INDEX -

Les fonctions de l'INDEX qui vont être décrites sont de type AUTOLOAD (i.e. il n'est pas besion de charger le fichier qui les contient, le système le fera pour vous automatiquement au premier appel de l'une de ces fonctions).
D'ordinaire ces fonctions se trouvent dans le fichier :
(SYS (INDEX . VLI)) pour la version interprétée
(SYS (INDEX . VLA)) pour la version compilée lisible
(SYS (INDEX . VLO)) pour la version compilée chargeable

(INDEXFILE <filout> <filin> <sw1>) [SUBR à 3 arguments]

traite toutes les fonctions incluses dans le fichier d'entrée <filin> et édite l'index de ces fonctions dans le fichier de sortie <filout>. Si l'indicateur <sw1>=T les fonctions apparaitront triées par ordre alphabétique dans le fichier de sortie; si l'indicateur <sw1>=NIL, les fonctions apparaitront dans le fichier de sortie dans le même ordre que dans le fichier d'entrée. S'il n'y a pas de spécification pour le fichier de sortie (i.e. si <filout> = NIL), le fichier (DSK (INDEX . VLX) () \055) est utilisé. L'extension standard des fichiers de sortie de l'INDEX est .VLX .
INDEXFILE ramène en valeur le nom du fichier de sortie utilisé.

---

(1) Ce programme a été conçu et réalisé par Patrick GREUSSAY.

(INDEXF <file> <sw1>) [FSUBR]

est une forme abrégée de la fonction INDEXFILE. L'appel (INDEXF <file> sw1>) correspond à l'appel :

```
    (INDEXFILE '(DSK (<file> . VLX) (GETPPN) \055)
               '(DSK (<file> . VLI))
               <sw1>)
```

Cette fonction ramène la valeur ramenée par INDEXFILE i.e. le nom du fichier de sortie utilisé.

10.3.2 Exemple De Sortie Obtenue Par L'Index -

(extrait de l'INDEX du fichier PRETTY.VLI)

```
                 .....
                 .....
      ---------- CROSSFILE ----------

TYPE = EXPR
ARGS = (FILOUT FILIN ?CROSSALL ?INDEX  NBLINE  NBFNT)
FVARS = (PRETTYSIZE)
FVARSET = (PRETTYSIZE)


      ---------- CROSSPRINT ----------

TYPE = EXPR
ARGS = ( NB X)
FVARS = ( NBFNT ?INDEX)
FVARSET = (?INDEX)
STRINGS = ("Cross Reference")
USEDBY = (CROSS EOF)

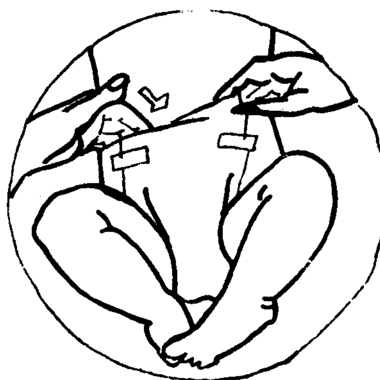

      ---------- EOF ----------

TYPE = EXPR
FVARS = (?INDEX FILIN FILOUT PTVRG)
USING = (%PC &EOF CROSSPRINT PRETTYFIN)


      ---------- FLATSIZE ----------

TYPE = EXPR
ARGS = (%L %N)
USING = (ADDN FLATSIZE)
USEDBY = (FLATSIZE SARENTRE)

                 .....
                 .....
                 .....
```



CHAPITRE 11

LE LAP ET LE LAPACK

Le LAP est un assembleur (ressemblant à l'assembleur du PDP10), conçu pour être utilisé par VLISP 10.3 et prévu originellement pour charger le code issu du compilateur. Il peut être utilisé seul pour écrire de nouvelles fonctions standard. Un minimum de connaissances de l'assembleur PDP10 et de l'organisation de l'interprète est requis pour utiliser le LAP sans dégats.
LAP reçoit en argument une liste dont les éléments peuvent être :
- des atomes litéraux (qui servent d'étiquettes)
- des listes représentant des instructions normales, des pseudo-instructions, ou des macro-LAP (MACLAP).
Cette liste peut être remplacée par un fichier s'il y a beaucoup d'instructions à charger.

Après assemblage ces instructions sont chargées en mémoire dans une zone code prévue à cet effet. Si cette zone se révèle trop petite une erreur apparait; le libellé de cette erreur est :
** no room for code.
Cette erreur est fatale. Il faut augmenter la taille de la zone allouée au code (dans la fonction CONFIGURATION du fichier initial CONFIG.INI) et relancer tout le chargement.

11.1 Les Registres

L'interprète utilise les 16 registres de la machine de la manière suivante :

no	symbole	remarques
00	RG	est le registre général lui-même. On peut donc tester directement au moyen d'un masque les bits utiles.
01	A1	accumulateur 1 ! Seuls ces 4 registres sont nettoyés par le G.C.
02	A2	accumulateur 2 !
03	A3	accumulateur 3 !
04	A4	accumulateur 4 !
05	A5	accumulateur 5 — Ces 4 registres font office de registres auxiliaires.
06	A6	accumulateur 6
07	A7	accumulateur 7
10	A8	accumulateur 8
11	U1	user 1 ! Ces 2 registres ne sont jamais utilisés par l'interprète et sont donc réservés pour l'utilisateur.
12	U2	user 2 !
13	L	link — est utilisé pour les appels de S.P de type JSP.

14 STRG contient toujours le pointeur sur la liste libre des chaînes.

15 NUMB contient toujours le pointeur sur la liste libre des nombres.

16 FREE contient toujours le pointeur sur la liste libre des listes.

17 P est le pointeur sur l'unique pile utilisée par l'interprète.

11.2 Format Externe D'une Instruction

Chaque instruction est représentée par une liste de la forme :

(<codop> <registre> @ <adr> <index>)

<codop> - est le mnémonique de l'instruction (tous les mnémoniques des instructions PDP10 sont disponibles en revanche les mnémoniques des CALLI et des TTCALL ne le sont pas).

<registre> - est le numéro ou le symbole du registre 1er opérande.

@ - si cet atome est présent (et à cette position), le bit d'indirection de l'instruction sera positionné.

<adr> - représente la valeur du champ adresse qui sera chargée dans les 18 bits de poids faibles de l'instruction (la description de ce champ suit).

<index> - est le numéro ou le symbole du registe d'index. Ce dernier champ est optionnel.

le champ <adr> peut avoir la représentation suivante :

- un nombre (utilisé dans les intructions possédant des valeurs immédiates).
- un atome littéral qui peut être :
  - un symbole de registre
  - une étiquette
  - un symbole spécial. LAP connait en effet certaines adresses utiles de l'interprète. Ces adresses ont été chargées par le LINK-10 . Elles sont représentées par un symbole dont le 1er caractère est toujours un " : ".
- une liste représentant :
  - (nombre) une adresse relative au début de chargement.
  - (* nombre) une adresse relative au compteur d'assemblage.
  - (QUOTE objet lisp) ou ' objet lisp , l'adresse de l'objet lisp spécifié Cette adresse est en réalité l'index de cet objet par rapport au début de la zone des objets lisp.
  - (+ <adr1> ... <adrN>) la somme des différentes adresses spécifiées.
  - (:MEM <adr>) cet opérande est la forme abrégée de : (+ :MEM <adr>).

exemples d'instructions :

```
(MOVEI A1 2)
(MOVE A1 @ TAB A5)
(CAML A4 :BLIST)
(JRST 0 (* -5))
(CAIE A4 'LAMBDA)
(HRRZ A3 (+ :MEM 'X))
```

11.3 Les Pseudo-instructions

LAP connait un certain nombre de pseudos-instructions dont la forme externe est identique à celle des instructions normales.

(COMMENT ...) ou (* ...)
cette pseudo-instruction est complètement ignorée du LAP. Elle permet d'insérer des commentaires à l'intérieur d'un programme LAP.

(END)
indique la fin d'un assemblage (un 2ème peut suivre dans la liste des instructions fournie au LAP). Cette pseudo est automatiquement générée en fin de liste (ou de fichier) d'instructions ; elle n'est donc pas obligatoire.

(ENTRY <nom> <type> <nombre>)
permet de définir le point d'entrée d'une fonction qui devient une fonction standard de VLISP. Le nom de la fonction <nom> doit être un atome littéral, le type de cette fonction peut être SUBR ou FSUBR. Si le type est SUBR on peut spécifier son nombre d'arguments (si cette fonction possède un nombre d'arguments plus petit ou égal à 3).
ex : (ENTRY FACT SUBR 1)

(EVAL <s>)
évalue l'expression <s> spécifiée en argument. Cette pseudo-instruction ne charge aucun mot mémoire mais permet d'accéder directement à l'interprète.
ex: (EVAL (PRINT "2ème partie."))

(EXP <adr>)
réserve un mot qui est initialisé avec la valeur de <adr>.
ex : (EXP -1)

(OPCD <symbole> <valeur>)
permet de définir de nouveaux mnémoniques pour les codes instructions.
ex : (OPCD PJRST \254)

(QUOTE <s>) ou bien ' <s>
réserve un mot contenant un pointeur sur l'objet lisp <s> spécifié.

(REGISTER <symbole> <valeur>)
permet de définir de nouveaux mnémoniques pour les registres. L'argument <symbole> doit être un atome littéral, et sa valeur associée <valeur> doit être comprise dans l'intervalle [0 , 15].
ex : (REGISTER PILE \17)

(VALAP <symbole> <valeur>)
permet de définir de nouveaux symboles dont la valeur est donnée explicitement. L'argument <symbole> doit être un atome littéral et sa valeur <valeur> numérique.
ex : (VALAP :JBSYM \116)

(XWD <adr> <adr>) abreviation [<adr> <adr>]
réserve un mot dont les 2 demis-mots sont initialisés avec les valeurs des opérandes fournis.
ex : (XWD -1 (+ :MEM (* 3)))

11.4 Les Macro-LAP

Il est possible de définir des macros en utilisant la pseudo suivante :

(MACLAP <nom> <liste d'argument> <corps de la fonction>)

cette pseudo-instruction permet de définir les MACLAP . A l'apparition d'une intruction de la forme (<nom> <arg1> .. <argN>) dans laquelle <nom> est le nom d'une MACLAP, la fonction spécifiée est appellée avec <arg1> ... <argN> comme arguments. Si la valeur ramenée par cette application est une liste, elle est re-assemblée; si la valeur est NIL, rien n'est assemblé.

exemple de définition de MACLAP :

```
(MACLAP INCR (ATOM)
   [['HLRZ 1 ['+ ':MEM [QUOTE ATOM]]]
    '(PUSHJ P ADD1)
    ['HRLM 1 ['+ ':MEM [QUOTE ATOM]]]])
```

```
l'appel de   (INCR X)
sera expansé en   (HLRZ 1 (+ :MEM 'X))
                  (PUSHJ P ADD1)
                  (HRLM 1 (+ :MEM 'X))
```

Les MACLAP sont très puissantes car elles donnent accès à l'interprète lui-même et permettent de définir des pseudo-instructions très sophistiquées. Ces MACRO sont toujours traitées en premier lieu; on peut donc redéfinir toutes les instructions ainsi que les pseudo-instructions standard.

Il existe un certain nombre de MACLAP prédéfinies qui possèdent les définitions suivantes :

```
CAR          (MACLAP CAR (D S) [['HLRZ D ':MEM S]])

CDR          (MACLAP CDR (D S) [['HRRZ D ':MEM S]])

JPLIST       (MACLAP JPLIST (R A)
                     [['CAMGE R ':BLIST]
                      ['JRST 0 A]])

JNLIST       (MACLAP JNLIST (R A)
                     [['CAML R ':BLIST]
                      ['JRST 0 A]])

CONS         (MACLAP CONS (R)
                     [['EXCH R ':MEM 'FREE]
                      ['EXCH 'FREE R]])

UNCONS       (MACLAP UNCONS (R CAR CDR)
                (IF (NEQ R CDR)
                     [['HRRZ CDR ':MEM R]
                      ['HLRZ CAR ':MEM R]]
                     [['HLRZ CAR ':MEM R]
                      ['HRRZ CDR ':MEM R]]))
```

11.5 Accès Aux Objets VLISP

Les atomes littéraux, les nombres, les chaînes et les listes sont stockés dans des zones fixes du LOWSEG (cf: le chapitre Organisation et utilisation). Ces objets sont toujours représentés d'une manière interne par un POINTEUR sur ces zones ; ce pointeur n'est pas une adresse physique mais un index par rapport au début de cette zone. On a accès aux limites de ces zones au moyen de symboles spéciaux (dont le 1er caractère est un ":") qui sont connus du LAP.

:MEM adresse physique du début de la zone des objets VLISP.

- ZONE DES ATOMES LITERAUX -

:BNUMB index du début de la zone des nombres.

- ZONE DES NOMBRES -

:BSTRG index du début de la zone des chaînes.

- ZONE DES CHAINES -

:BLIST index du début de la zone des listes.

- ZONE DES LISTES -

11.6 Test De Type

Grâce à ce découpage de la mémoire, le test de type se ramène à une comparaison avec les limites des zones, ce qui est très efficient. Le 1er atome stocké dans la zone des atomes littéraux est l'atome NIL ; la valeur de son index est 0 ; les tests par-rapport à NIL peuvent se coder en une seule instruction (JUMPE ou JUMPN).

ex :

```
; branchement si A1 est égal à NIL ;
(JUMPE A1 quelquepart)

; branchement si A2 n'est pas une liste ;
(CAML A1 :BLIST)
(JRST 0 quelquepart)

; branchement si A1 est un nombre ;
(CAML A1 :BNUMB)
(CAML A1 :BSTRG)
(JRST 0 quelquepart)
```

11.7 Création D'objets

Pour créer un atome littéral, il faut préparer dans un petit buffer (de 3 mots) de nom :PNAME (ce nom est connu de LAP) le P-name de l'atome que l'on veut créer puis appeller une routine de l'interprète de nom :TRYATOM au moyen d'un (PUSHJ P :TRYATOM). Ce buffer doit contenir les caractères de P-name en code ASCII. Le 1er caractère doit être le nombre total de caractères du P-name. Ce buffer est complété avec des 0. Au retour le registre A1 contient le pointeur sur ce nouvel atome.

Pour créer un nombre, il faut mettre sa valeur dans le registre A5 puis appeller la routine :CRANUM pour créer un nombre entier ou bien la routine :CRAFLT pour un nombre flottant au moyen d'un (PUSHJ P :CRANUM / :CRAFLT). Au retour le registre A1 contient le pointeur sur ce nouveau nombre.

Pour créer une chaîne, il faut mettre la liste contenant tous les caractères (sous forme atomique) dans le registre A1 puis appeller la routine de l'interprète :CRASTR au moyen d'un (PUSHJ P :CRASTR). Au retour le registre A1 contient le pointeur sur cette nouvelle chaîne.

On peut créer un doublet de liste directement. Il faut préparer un registre <r> quelconque (A1 A2 A3 ou A4) avec en partie gauche le CAR du doublet et en partie droite son CDR puis stocker le doublet et actualiser FREE. Il n'est pas nécéssaire de tester la fin de la liste libre. En effet une adresse invalide, mise en fin de liste libre, provoque une interruption si on essaie d'accèder à ce mot, interruption qui est reconnue comme tentative de création de doublet et qui lance automatiquement le garbage-collecting. Il est impératif d'utiliser les deux instructions suivantes et sous ce format pour que le G.C. puisse être invoqué automatiquement.

```
          ; préparation du registre <r> puis
     (EXCH <r> :MEM FREE)        ; création du doublet
     (EXCH FREE <r>)             ; actualisation de FREE
          ; le pointeur obtenu se trouve dans <r>
```

11.8 Appel Des Fonctions

Toutes les fonctions de l'interprète sont appellées au moyen d'un
PUSHJ P à, l'adresse de lancement de la fonction.
Toutes les fonctions doivent donc se terminer par un POPJ P. Les arguments des fonctions sont transmis dans différents registres en fonction du type de la fonction. Cette transmission est effectuée automatiquement, avant l'appel des fonctions, par les fonctions interprète EVAL ou APPLY. Quelque soit le type d'une fonction, sa valeur est toujours retournée dans le registre A1 (il faut donc que ce registre soit chargé avant le retour de la fonction). Les fonctions de type SUBR à 1 argument reçoivent leur argument évalué dans le registre A1, les SUBR à 2 arguments reçoivent leurs arguments évalués dans les registrès A1 (pour le 1er) et A2 (pour le 2ème), les SUBR à 3 arguments reçoivent leurs argments évalués dans les registres A1 (pour le 1er) A2 (pour le 2ème) et A3 (pour le 3ème), les SUBR à nombre quelconque d'arguments reçoivent une liste contenant tous les arguments évalués dans le registre A4, les fonctions de type FSUBR reçoivent la liste des arguments non-évalués dans le registre A1.

11.9 Les Fonctions Standard Utilisées Par Le LAP

Le LAP utilise un certain nombre de fonctions spéciales de l'interprète. Ces fonctions sont tout naturellement disponibles.

(GETSYMBOL <at>) [SUBR à 1 argument]

ramène la valeur du symbole atomique <at>. Cet atome littéral doit commencer par le caractère spécial : et avoir moins de 7 caractères au total, donc être de la forme :xxxxxx . Si l'atome <at> ne possède pas de valeur, GETSYMBOL ramène NIL. Cette fonction permet d'avoir accès à la table des symboles, fabriquée par le LINK-10, qui contient tous les symboles globaux de l'interprète VLISP 10.3.

ex : (GETSYMBOL ':GARBCL) ☞ 1220 (par exemple)

(GETSYMBOL ':XYZ)   ☞   NIL

(OPCD <at>) [SUBR à 1 argument]

ramène le code opération de l'instruction machine de nom <at>. Si <at> n'est pas un mnémonique d'une instruction connue, OPCD ramène NIL.

ex : (OPCD 'JRST) ☞ 254
     (OPCD 'PJRST) ☞ NIL

(REGISTER <at>) [SUBR à 1 argument]

ramène la valeur de l'atome <at> considéré comme le nom d'un registre. Cette fonction connait le nom des registres standard. Si <at> n'est pas le nom d'un registre standard, REGISTER ramène NIL.

ex : (REGISTER 'A5) ☞ 5
     (REGISTER 'A9) ☞ NIL

11.10 Les Fonctions Standard Du LAP

Les fonctions du LAP sont de type AUTOLOAD (i.e. il n'est pas besoin de charger le fichier qui les contient, le système le fera automatiquement pour vous au premier appel de l'une de ces fonctions).
D'ordinaire ces fonctions se trouvent dans le fichier :
(SYS (LODLAP . VLI)) pour la version interprétée
(SYS (LODLAP . VLA)) pour la version compilée lisible
(SYS (LODLAP . VLO)) pour la version compilée chargeable

(LAP <l> <swl>) [SUBR à 2 arguments]

est la principale fonction du LAP. Le premier argument <l> doit être une liste d'instructions LAP; le deuxième argument <swl> est un indicateur qui vaut T si vous désirez un listage de l'assemblage. Le LAP étant un assembleur "une passe", les adresses des références avant ne sont pas correctes (et valent toujours 0).

(LAPFILE <filout> <filin> <swl>) [SUBR à 3 arguments]

le fichier d'entrée <filin> est un fichier qui ne doit contenir que des instructions LAP. LAPFILE va assembler et charger ces instructions. Le fichier de sortie <filout> contiendra tous les points d'entrée du fichier chargé, ainsi que les erreurs le cas échéant. Si l'indicateur <swl> = T un listage de l'assemblage sera produit également sur le fichier de sortie <filout>. Comme pour la fonction LAP, l'assembleur n'exécutant qu'une passe, les adresses des "références avant" ne seront pas correctes (et auront toujours la valeur 0). Si le fichier de sortie <filout> n'est pas fourni (i.e. si <filout> = NIL), le fichier (DSK (LODLAP . LOD) (GETPPN) \055) sera utilisé. L'extension standard des fichiers de sortie de LAPFILE est .LOD . Si le fichier d'entrée <filin> n'est pas fourni (i.e. si <filin> = NIL) le fichier (DSK (LODLAP . VLA) (GETPPN)) sera utilisé. L'extension standard des fichiers d'entrée de LAPFILE est .VLA .
LAPFILE ramène en valeur le nom du fichier de sortie utilisé.

(LAPF <file> <sw1>) [FSUBR]

est une forme abrégée de la fonction LAPFILE. L'appel (LAPF <file> <sw1>) correspond à l'appel suivant :

```
(LAPFILE '(DSK (<file> . LOD) (GETPPN) \055)
         '(DSK (<file> . VLA))
         <sw1>)
```

LAPF permet donc de charger un fichier d'intructions LAP d'extension .VLA comme par exemple des fichiers issus du compilateur.

(LAPEND) [SUBR à 0 argument]

permet de récupérer la place occupée par les fonctions du LAP, ainsi que les indicateurs placés sur certains atomes par le LAP. Si certains atomes contiennent toujours des références non résolues (ces atomes possèdent sur leur P-liste l'indicateur *UDS), un avertissement est donné sous la forme :

```
** undefined symbol : le nom de la fonction ou
                      de l'etiquette toujours inconnue.
```

11.11 Les Erreurs Détectées Par Le LAP

Durant l'assemblage, un certain nombre d'erreurs sont détectées par le LAP. Ces erreurs ne sont jamais fatales mais le code généré est bien évidemment faux. Le libellé d'erreur est le suivant :

** LAP error : <type> in <arg>

dans lequel le type de l'erreur <type> est imprimé ainsi que l'argument défectueux.

Les différents types d'erreur sont :

ADRESS - l'opérande de type <adr> est erronné.

GETSYMBOL - un symbole commence par le caractère spécial : mais n'est pas connu de la fonction GETSYMBOL et n'a pas été déclaré au moyen de la pseudo-instruction VALAP.

LOD ou LODMEM - instruction inconnue.

OPCD - le champ <opcd> d'une instruction est mal codé ou bien la pseudo-instruction OPCD est mal employée.

REG - un des champs <registre> ou <index> d'une instruction est incorrect.

REGISTER - mauvaise utilisation de la pseudo-instruction REGISTER.

11.12 Exemples D'utilisation Du LAP

```
?
?  ; DEFINITION DE LA FONCTION ONEP                   ;
?  ; QUI POSSEDE EN VLISP LA DEFINITION SUIVANTE :    ;
?
?  ; (DE ONEP (X) (IF (EQ X 1) T NIL))                ;
?
?  (LAP '((ENTRY ONEP SUBR 1)
?         (CAIE A1 '1)
?         (TDZA A1 A1)
?         (MOVEI A1 'T)
```

```
?          (POPJ P))
?     T)

   107206                          (ENTRY ONEP SUBR 1)
   107206 302 1  0 0  13761        (CAIE A1 '1)
   107207 634 1  0 0  1            (TDZA A1 A1)
   107210 201 1  0 0  6            (MOVEI A1 'T)
   107211 263 17 0 0  0            (POPJ P)
   107212

   (107206 ONEP SUBR 1)

                                   (END)
=  LAP
=  ; time = 160 ms ;
?  ?
?  (TYPEFN 'ONEP)
=  SUBR
=  ; time = 0 ms ;
?
?  (ONEP 1)
=  T
=  ; time = 0 ms ;
?
?  (ONEP)
=  NIL
=  ; time = 0 ms ;
?
?  ; DEFINITION DES FONCTIONS              ;
?  ; (DE CADDDAR (X) (CAR (CDDDAR X)))     ;
?  ; (DE CADDDDR (X) (CDR (CDDDAR X)))     ;
?
?  (LAP '((ENTRY CADDDDR SUBR 1)
?         (SKIPA A1 :MEM A1)
?         (ENTRY CADDDAR SUBR 1)
?         (HLRZ  A1 :MEM A1)
?         (JRST  0 CADDDR))
?     T)

   107212                          (ENTRY CADDDDR SUBR 1)
   107212 334 1  0 1  2336         (SKIPA A1 :MEM A1)
   107213                          (ENTRY CADDDAR SUBR 1)
   107213 554 1  0 1  2336         (HLRZ A1 :MEM A1)
   107214 254 0  0 0  407417       (JRST 0 CADDDR)
   107215

   (107212 CADDDDR SUBR 1)
   (107213 CADDDAR SUBR 1)
                                   (END)
=  LAP
=  ; time = 180 ms ;
?
?  (CADDDAR '((A B C D E) F G H I J))
=  D
=  ; time = 0 ms ;
?  (CADDDDR '((A B C D E) F G H I J))
=  I
=  ; time = 0 ms ;
?
?  ; REDEFINITION DE LA FONCTION REVERSE STANDARD ;
?  ; (DE REV (L1 L2)                               ;
?  ;   (WHILE (LISTP L1)                           ;
?  ;     (SETQ L2 (CONS (CAR L1) L2)               ;
```

```
? ;           L1 (CDR L1))))                         ;
?
? (LAP '((* "REVERSE très STANDARD.")
?       RE
?         (HLL 2 :MEM 1)
?         (CONS 2)
?         (CDR 1 1)
?         (ENTRY REV SUBR 2)
?         (JPLIST 1 RE)
?         (MOVEI 1 0 2)
?         (POPJ P))
?    T)

  107215                         (* "REVERSE très STANDARD.")
  107215                         RE
  107215 500 2  0 1  2336        (HLL 2 :MEM 1)
  107216 250 26 0 16 2336        (EXCH 2 :MEM FREE)
  107217 250 16 0 0  2           (EXCH FREE 2)
  107220 550 10 0 1  2336        (HRRZ 1 :MEM 1)
  107221                         (ENTRY REV SUBR 2)
  107221 311 11 0 0  1231        (CAML 1 :BLIST)
  107222 254 0  0 0  107215      (JRST 0 RE)
  107223 201 1  0 2  0           (MOVEI 1 0 2)
  107224 263 17 0 0  0           (POPJ P)
  107225

  (107221 REV SUBR 2)
                                 (END)
= LAP
= ; time = 320 ms ;

?
? (REV '(A (B C) D . E) '(F G H))
= (D (B C) A F G H)
= ; time = 0 ms ;
?
```

11.13 Le LAPACK

Les fichiers contenant du LAP (d'extension .VLA) sont très lisibles mais sont volumineux et longs à charger. Les fonctions du LAPACK vont tasser des instructions LAP en effectuant une première passe qui va résoudre toutes les références absolues, les noms de registres, les codes instructions et d'une manière générale tout ce qui n'est pas relocatable. Le résultat en est une nouvelle liste (ou un nouveau fichier) d'instructions LAP "tassées", chargeable plus rapidement mais devenues "illisibles".

Les fonctions du LAPACK sont de type AUTOLOAD (i.e. il n'est pas besoin de charger le fichier qui les contient, le système le fera pour vous automatiquement au premier appel de l'une de ces fonctions.
D'ordinaire ces fonctions se trouvent dans le fichier :
    (SYS (LAPACK . VLI)) pour la version interprétée
    (SYS (LAPACK . VLA)) pour la version compilée lisible
    (SYS (LAPACK . VLO)) pour la version chargeable

(LAPACK <l>) [SUBR à 1 argument]

> tasse la liste d'instructions LAP <l>, et ramène une nouvelle liste d'instructions tassées.

(LAPACKFILE <filout> <filin>) [SUBR à 2 arguments]

Le fichier <filin> est un fichier qui ne contient que des instructions LAP; LAPACKFILE va créer un nouveau fichier <filout> qui contiendra toutes les intructions du fichier d'entrée <filin> en format tassé. Ce nouveau fichier est bien entendu chargeable par le LAP.
Si le fichier de sortie <filout> n'est pas fourni (i.e. si <filout> = NIL) alors le fichier (DSK (LAPACK . VLO) (GETPPN) \055) est utilisé. L'extension standard des fichiers de sortie du LAPACKFILE est .VLO . Si le fichier d'entrée <filin> n'est pas fourni (i.e. si <filin> = NIL) alors le fichier (DSK (LAPACK . VLA) (GETPPN)) est utilisé. L'extension standard des fichiers d'entrée du LAPACKFILE est .VLA .
LAPACKFILE ramène en valeur le nom du fichier de sortie utilisé.

(LAPACKF <file> <sw1>) [FSUBR]

est une forme abrégée de la fonction LAPACKFILE. L'appel (LAPACKF <file> <sw1>) correspond à l'appel suivant :

```
   (LAPACKFILE '(DSK (<file> . VLO) (GETPPN) \055)
               '(DSK (<file> . VLA))
               <sw1>)
```

LAPACKF permet donc de tasser un fichier d'intructions LAP d'extension .VLA comme par exemple des fichiers issus du compilateur.

11.14 Exemples D'utlisation Du LAPACK

```
?
?  ; L contient le code de la fonction KWOTE  ;
?  ; qui peut se définir en VLISP : ;
?
?  ;   (DE KWOTE (l)          ;
?  ;     (IF (ATOM l)         ;
?  ;         ['QUOTE l]       ;
?  ;         l))              ;
?
?  (SETQQ L
?      ((ENTRY KWOTE SUBR 1)
?          (CAML 1 :BLIST)
?          (POPJ P)
?          (HRLZ 1 1)
?          (EXCH 1 :MEM FREE)
?          (EXCH FREE 1)
?          (HRLI 1 'QUOTE)
?          (EXCH 1 :MEM FREE)
?          (EXCH FREE 1)
?          (POPJ P)))
=  ((ENTRY KWOTE SUBR 1) (CAML 1 :BLIST) (POPJ P)
= (HRLZ 1 1) (EXCH 1 :MEM FREE) (EXCH FREE 1)
= (HRLI 1 'QUOTE) (EXCH 1 :MEM FREE) (EXCH FREE 1)
= (POPJ P))
=  ; time = 0 ms ;
?
?  (LAPACK L)
   (ENTRY KWOTE SUBR 1) 26986152601 24150802432
  -24150802431 22560638174 22666018817
  -25090326510 22560638174 22666018817
  24150802432
=  ; time = 60 ms ;
?
```

CHAPITRE 12

LE COMPILATEUR

VLISP 10.3 possède un compilateur. Il est sans danger et peut être utilisé par tous. Ce compilateur peut traduire n'importe quelle fonction définie de type EXPR ou FEXPR en une suite d'instructions qui seront assemblées et chargées par le LAP. Il n'utilise AUCUNE déclaration spéciale et suppose que les fonctions que vous voulez compiler sont CORRECTES. Les fonctions ainsi compilées sont exécutées entre 3 et 10 fois plus rapidement, et occupent moins de place (avec un gain variant entre 20 et 200 %). Ce compilateur garantit l'identité de résultat avec la forme interprétée sauf en cas d'automodification de fonction ou en cas de redéfinition dynamique de fonctions standard ou en cas d'évaluation symbolique.

12.1 Les Macro Du Compilateur

Le compilateur utilise un certains nombre de macro-LAP, connues du LAP qui facilite son écriture et font gagner de la place à la génération du code.

Toutes ces macro ne chargent qu'un seul mot mémoire.

(ARRRAY <reg> <atome>)

met dans le registre spécifié <reg> l'adresse du descripteur du tableau <atome>. Cette macro est expansée en :
(HRRZ <reg> [+ :MEM 5 '<atome>])

(CAR <regd> <source>)

met le CAR de l'objet <source> dans le registre <regd>. Cette macro est expansée en :
si l'objet <source> est
- un registre (HLRZ <regd> :MEM <source>)
- un atome (HLRZ <regd> [+ :MEM 'atome])
cette forme est équivalente au GETVAL.

(CDR <regd> <source>)

met le CDR de l'objet <source> dans le registre <regd>. Cette macro est expansée en :
si l'objet <source> est
- un registre (HRRZ <regd> :MEM registre)
- un atome (HRRZ <regd> [+ :MEM 'atome])

(GETVAL <registre> <atome>)

charge dans le registre spécifié la C-valeur de l'atome. Cette macro est expansée en :
(HLRZ <registre> [+ :MEM '<atome>])

(PUTVAL <registre> <atome>)

met dans la C-valeur de l'atome spécifié, la valeur du registre. Cette macro est expansée en :
(HRLM <registre> [+ :MEM '<atome>])

(RPLACA <regs> <dest>)

met le registre <regs> dans le CAR de l'objet <dest>. Cette macro est expansée en :
si l'objet <dest> est
- un registre (HRLM <regs> :MEM <dest>)
- un atome (HRLM <regs> [+ :MEM 'atome])

(RPLACD <regs> <dest>)

met le registre <regs> dans le CDR de l'objet <dest>. Cette macro est expansée en :
si l'objet <dest> est
- un registre (HRRM <regs> :MEM <dest>)
- un atome (HRRM <regs> [+ :MEM 'atome])

(SETNIL <atome>)

met NIL dans la C-valeur de l'atome spécifié. Cette macro est expansée en :
(HRRZS 0 [+ :MEM '<atome>])

12.2 Points D'entrée Spéciaux

Le compilateur utilise un certain nombre de points d'entrée spéciaux dans l'interprète. Les noms de ces points d'entrée sont toujours préfixés par le caractère " : " et sont bien évidemment connus de LAP.

12.2.1 Retour Standard De Fonction -

:VPOPJ
est l'adresse d'une instruction (POPJ P)

:TRUTH
est l'adresse des retours vrais des prédicats. Elle correspond aux deux instructions : (MOVEI 1 'T) (POPJ P).

:FALSE
est l'adresse des retours faux des prédicats. Elle correspond aux deux instructions : (SETZ 1) (POPJ P).

:CRAZER
est l'adresse de création du nombre VLISP 0. Elle correspond aux deux instructions : (MOVEI 1 '0) (POPJ P).

:CRAONE
est l'adresse de création du nombre VLISP 1. Elle correspond aux deux instructions : (MOVEI 1 '1) (POPJ P).

12.2.2 Création De Nombres -

:CRANUM
crée le nombre VLISP dont la valeur se trouve dans le registre A5; au retour de ce sous-programme, A1 contient un pointeur sur le nouveau nombre créé. Cette routine se termine par un (POPJ P), il faut donc l'appeller par (PUSHJ P :CRANUM). Le PUSHJ étant une instruction très lente, le compilateur utilise de préférence le point d'entrée suivant et ne se sert de :CRANUM qu'en cas de "JRST hack".

:$CRANB
est identique au point d'entrée précédent mais est appellé par (JSP L :$CRANB) ce qui est plus rapide. Cette routine se termine donc par un (JRST 0 0 L).

:$CRANP

l'appel	correspond aux deux instructions
(JSP L :$CRANP)	(JSP L :$CRANB) (PUSH P A1)

ce point d'entrée ne sert donc qu'à réduire le code produit par le compilateur qui doit souvent empiler les nombres qui viennent d'être créés.

12.2.3 Points D'entrée Spéciaux Des SUBR -

Le lancement des NSUBR étant un dispositif plutot lourd (voir le point d'entrée :NSUBR), le compilateur utilise des points d'entrée spéciaux pour les NSUBR courantes qui ne possèdent qu'un ou deux arguments, ce qui évite de créer des listes d'arguments évalués.

Le compilateur dans d'autres cas va "casser" des appels de type NSUBR en une suite d'appels de 1SUBR. Par exemple la forme (PRIN1 A B C) sera compilée (PROGN (:$PRIN1 A) (:$PRIN1 B) (:$PRIN1 C)), :$PRIN1 étant le point d'entrée de la fonction PRIN1 à un argument évalué.

En outre, des points d'entrée spéciaux ont également été créés pour des fonctions qui, possédant un nombre fixe d'arguments, n'en utilisait en général qu'un nombre réduit.

:$1STATUS
point d'entrée de la fonction STATUS qui ne possède qu'un argument (:$1STATUS est considérée donc comme une SUBR à 1 argument).

:$2STATUS
point d'entrée de la fonction STATUS qui ne possède que deux arguments (:$2STATUS est considérée donc comme une SUBR à 2 arguments).

:$3STATUS
point d'entrée de la fonction STATUS qui ne possède que trois arguments (:$3STATUS est considérée donc comme une SUBR à 3 arguments).

:$PLUS
est l'adresse de la fonction PLUS à deux arguments qui doivent se trouver respectivement dans les registres 1 et 2. Ce point d'entrée est utilisé quand la fonction PLUS n'est pas utilisée en tant que NSUBR.

:$DIFFER
est l'équivalent de l'adresse :$PLUS, pour la NSUBR DIFFER.

:$TIMES
est l'équivalent de l'adresse :$PLUS, pour la NSUBR TIMES.

:$QUO
est l'équivalent de l'adresse :$PLUS, pour la NSUBR QUO.

:$REM
est l'équivalent de l'adresse :$PLUS, pour la NSUBR REM.

:$MAX
est l'équivalent de l'adresse :$PLUS, pour la NSUBR MAX.

:$MIN
est l'équivalent de l'adresse :$PLUS, pour la NSUBR MIN.

:$GT
est l'équivalent de l'adresse :$PLUS, pour la NSUBR GT.

:$GE
est l'équivalent de l'adresse :$PLUS, pour la NSUBR GE.

:$LE
est l'équivalent de l'adresse :$PLUS, pour la NSUBR LE.

:$LT
est l'équivalent de l'adresse :$PLUS, pour la NSUBR LT.

:$PRIN1
est le point d'entrée de la fonction PRIN1 à 1 argument qui doit se trouver dans A1. Le compilateur utilise ce point d'entrée pour tous les appels de la fonction PRIN1 en macrogénérant s'il y a lieu une suite d'appels de la fonction :$PRIN1.
ex : l'appel (PRIN1 X (FOO L) '= (BAR L)) sera macrogénéré
```
(PROGN (:$PRIN1 X) (:$PRIN1 (FOO L)) (:$PRIN1 '=) (:$PRIN1 (BAR L)))
```

:$PRINT
est le point d'entrée de la fonction PRINT à 1 argument qui doit se trouver dans A1. Le compilateur utilise ce point d'entrée pour tous les appels de la fonction PRINT en macrogénérant s'il y a lieu une suite d'appels de la fonction :$PRIN1 et :$PRINT.
ex : l'appel (PRINT X (FOO L) '= (BAR L)) sera macrogénéré
```
(PROGN (:$PRIN1 X) (:$PRIN1 (FOO L)) (:$PRIN1 '=) (:$PRINT (BAR L)))
```

:$PUSH
est le point d'entrée de la fonction PUSH à 1 argument qui doit se trouver dans A1. Le compilateur utilise ce point d'entrée pour tous les appels de la fonction PUSH en macro-générant s'il y a lieu une suite d'appels de la fonction :$PUSH.
ex : l'appel (PUSH X Y Z) sera macro-généré
```
(PROGN (:$PUSH x) (:$PUSH Y) (:$PUSH Z))
```

:$POP
est le point d'entrée de la fonction POP sans argument. :$POP se comporte donc comme une SUBR sans argument.

:$TERPRI
est le point d'entrée de la fonction TERPRI sans argument. :$TERPRI se comporte donc comme une SUBR sans argument.

:$SPACES
est le point d'entrée de la fonction SPACES sans argument. :$SPACES se comporte donc comme une SUBR sans argument.

12.2.4 Fonctions Spéciales De L'interprète -

:$MAP1 :$MAPN :$MAPC1 :$MAPCN
ces points d'entrée sont utilisés pour exécuter les fonctions de type MAP ou MAPC sans passer par la fonction APPLY. Ils peuvent être utilisés quand le compilateur connait le type de la fonction à appliquer aux éléments successifs de la liste. Ces fonctions peuvent être soit des fonctions standards, soit des fonctions compilées de type FUNCTION. Si cette fonction est du type 1/2/3SUBR le point d'entrée utilisé est :$MAP1 ou :$MAPC1 sinon le compilateur utilise le point d'entrée :$MAPN ou :$MAPCN. L'exécution de ce type de MAP est fortement accéléré par cette possibilité. Ces points d'entrée attendent la liste dans le registre 1 et le nom de la fonction dans le registre 2.

Par exemple le point d'entrée :$MAPC1 correspond à la séquence :

```
$MAPC1 (HRRZ    A5 (+ :MEM 5) A2)  ; sauve l'adr. de lancement de la fonction.
       (PUSH    P A5)
       (PUSH    P A1)              ; sauve la liste d'argument.
```

```
        (JRST     0 SMAPC4)
SMAPC3  (UNCONS   A1 A1 A3)     ; élément suivant de A1.
        (MOVEM    A3 0 P)       ; sauve le reste.
        (SETZB    A2 A3)        ; pour les 2-3SUBR.
        (PUSHJ    P @ -1 P)     ; on lance la 1/2/3SUBR.
        (MOVE     A1 0 P)       ; puis on récupère le reste de la liste.
SMAPC4  (JPLIST   A1 SMAPC3)    ; ya encore des éléments.
        (SUB      P %T1)        ; dépile le reste et l'adr. de lancement.
        (POPJ     P)            ; c'est fini.
%T1     [2 2]                   ; pour repositionner la pile
```

:NSUBR

ce point d'entrée est utilisé pour lancer les NSUBR de l'interprète ou bien les vôtres. Les NSUBR demandent que leurs arguments, évalués, soient ressemblés en une liste se trouvant dans le registre A4. Ceci est fait automatiquement si la pile est chargée de la manière suivante : Il faut empiler d'abord un mot contenant dans sa partie gauche la valeur -1 et dans sa partie droite le nom de la NSUBR à lancer. Les arguments sont alors évalués un à un et leurs valeurs sont empilées sauf le dernier qui reste dans le registre 1. Puis le sous-programme :NSUBR est appellé via le registre L, au moyen d'un JSP.

ex: macrogénération de l'expression
(TIMES 7 (STATUS 4 3 2) 8)

```
    (PUSH  P :T1)       ; empile (XWD -1 TIMES) ;
    (PUSH  P :T2)       ; empile la valeur 7 ;
    (PUSH  P :T3)       ; empile (XWD -1 STATUS) ;
    (PUSH  P :T4)       ; empile la valeur 4 ;
    (PUSH  P :T5)       ; empile la valeur 3 ;
    (MOVEI 1 '2)        ; le dernier argument
                          de STATUS ;
    (JSP   L :NSUBR)    ; appel de STATUS ;
    (MOVEI 1 '8)        ; le dernier argument
                          de TIMES ;
    (JSP   L :NSUBR)    ; appel de TIMES ;
                        ; Table utilisée car le PDP-10
                          ne possède pas l'instruction
                          PUSH immédiat !! ;
:T1 (XWD  -1 TIMES)
:T2 '7
:T3 (XWD  -1 STATUS)
:T4 '4
:T5 '3
```

Ce genre d'appel est plutôt lourd et encombrant, le compilateur essaie d'éviter cela dans le cas des NSUBR utilisées fréquemment et qui n'ont qu'un ou deux arguments.

:NSUBRP

ce point d'entrée est équivalent à :NSUBR suivi d'un POPJ.

:SBIND1 :SBIND2 :SBIND3 :SBIND :FSBIND

ces points d'entrée permettent d'effectuer les liaisons dynamiques des arguments formels à l'entrée d'une fonction dans le cas où il faut préserver les environnements. Ces modules après avoir sauvé les anciennes valeurs des paramètres formels dans la pile et avoir lié les parametres actuels, vont empiler l'adresse du module qui déliera ces paramètres formels, ce qui permettra de retourner de ces fonctions de la manière habituelle, i.e. par un (POPJ P).

Il faut de plus fournir le nom de la fonction pour que ces modules puissent traiter les cas de tail-récursion et de co-tail-récursion dans les fonctions compilées qui lient les variables.
:SBIND1 est utilisé dans les SUBR à 1 argument, :SBIND2 dans les SUBR à 2 arguments, :SBIND3 dans les SUBR à 3 arguments, :SBIND dans les SUBR à N arguments et :FSBIND dans les FSUBR.
La forme des appels de ces modules est la suivante :

```
(JSP L :SBIND1/:SBIND2/:SBIND3/:SBIND/:FSBIND)
(XWD nom de la fonction liste d'arguments formels)
```

:ESBIND :ESCAPT

permet de gérer les appels des fonctions d'échappements. La définition d'un ESCAPE s'effectue au moyen de :

```
(JSP L :ESBIND)
(XWD étiquette de fin d'ESCAPE nom du ESCAPE)
```

Un appel d'une fonction d'échappement s'effectue :

```
(MOVEI A2 le nom du ESCAPE)
(JRST 0 :$ESCAPT)
```

12.3 Les Fonctions Standard Du Compilateur

Toutes les fonctions standard qui vont être décrites sont de type AUTOLOAD (i.e. il n'est pas nécessaire de charger le fichier qui les contient, le système le fera pour vous au premier appel de l'une de ces fonctions).
D'ordinaire elles se trouvent dans le fichier :

```
(SYS (COMPIL . VLI)) pour la version interprétée
(SYS (COMPIL . VLA)) pour la version compilée lisible
(SYS (COMPIL . VLO)) pour la version chargeable.
```

(COMPILES <s>) [EXPR à 1 argument]

crée une liste contenant les instructions LAP nécessaire à l'exécution de la S-expression <s>. COMPILES ramène cette liste en valeur. Cette fonction est principalement utilisée pour tester le fonctionnement du compilateur.

(COMPILE <at> <sw1> <sw2>) [FEXPR]

est la fonction de compilation en mode conversationnel. Cette fonction compile, assemble et charge une fonction quelconque. La fonction à compiler de nom <at> doit possèder une définition de type EXPR ou FEXPR. L'indicateur <sw1> est égal à T si vous voulez un listing du résultat de la compilation; l'indicateur sw2 sera transmis au LAP.

(COMPILEFILE <filout> <filin> <sw1>) [EXPR]

Cette fonction crée un fichier de sortie, de spécification <filout>, contenant la traduction sous forme d'appel de la fonction LAP, de toutes les définitions de fonctions contenues dans le fichier d'entrée de spécification <filin>. Le fichier ainsi obtenu pourra être relu par VLISP. Les S-expressions non compilables sont remises (au même endroit) dans le fichier de sortie.

(COMPILEF <file> <sw1>) [FEXPR]

est utilisée pour la compilation d'un fichier standard. (COMPILEF <file> <sw1>) est la forme abrégée de

```
(COMPILEFILE '(DSK (<file> . VLA))
             '(DSK (<file> . VLI))
             <sw1>)
```

(COMPILOPTIONS <i1> .... <iN>) [EXPR à N arguments]

permet de positionner certains indicateurs internes contrôlant le fonctionnement du compilateur. Les indicateurs sont DANS L'ORDRE :

- ?hacks : qui indique de valider la dernière passe d'amélioration locale du code produit par le compilateur. Cette dernière passe étant sans danger, on peut utiliser cet indicateur par tous temps; il n'est utilisé en principe que pour la maintenance du compilateur.

- ?ssec : qui permet d'effectuer la recherche des sous-expressions communes à l'interieur des fonctions compilées. Cette recherche étant fort longue et n'apportant que peu de résultats, il est conseillé de ne pas positionner cet indicateur.

- ?slonum : qui indique au compilateur d'utiliser les routines de l'interprète pour exécuter les fonctions arithmétiques. Cette option fait gagner de la place au niveau du code produit par le compilateur mais ralentit son exécution.

- ?open : qui indique de macrogénérer au maximum les fonctions standard et donc de ne pas passer par les routines de l'interpréte. Tout comme l'option précédente, la présence de cet indicateur va rallonger et accélérer le code généré.

- ?ckarray : indique qu'il faut passer par les routines de l'interprète pour calculer les indices des tableaux. Si cet indicateur n'est pas positionné, les indices sont calculés directement par le code généré et ne sont pas testés.

- ?filap : indique a la fonction COMPILEFILE qu'il faut produire un fichier de sortie d'extension .LAP (qui est l'ancien format des fichiers assembleurs). Cette option a été introduite pour des raisons de compatibilité et ne doit plus être utilisée.

```
; Voici la liste des OPTIONS standard ;

(COMPILOPTIONS
   T     ; ?hacks : utilisée pour tester le compilo. ;
   NIL   ; ?ssec : pour calculer les sous-exprs communes ;
   NIL   ; ?slonum : on utilise les nombres lents ;
   T     ; ?open : on macrogénère au maximum ;
   NIL   ; ?ckarray : on contrôle les indices des tableaux ;
   NIL   ; ?filap : on cré une file .LAP (sinon .VLA) ;
)
```

(COMPILEND) [EXPR à 0 argument]

récupère la place occupée par les fonctions du compilateur. COMPILEND remet les indicateurs AUTOLOAD des fonctions standard et ramène le nombre de doublets libérés en valeur. Cette fonction enlève de plus tous les indicateurs posés par le compilateur.

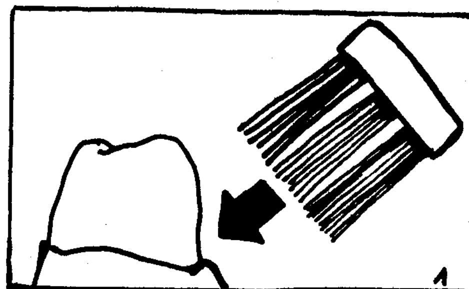
12.4 Exemples D'utilisation Du Compilateur

```
; 1 KWOTE-----------------------------------------------------

(DE KWOTE (L) (IF (LISTP L) L [QUOTE L]))

 FUNCTION LENGTH = 15
 #LAP LENGTH = 9
;
;;;;;;
          (ENTRY KWOTE SUBR 1)
          (CAML 1 :BLIST)
          (POPJ P)
 G101
          (HRLZ 1 1)
          (EXCH 1 :MEM FREE)
          (EXCH FREE 1)
          (HRLI 1 'QUOTE)
          (EXCH 1 :MEM FREE)
          (EXCH FREE 1)
          (POPJ P)

 (END)
```



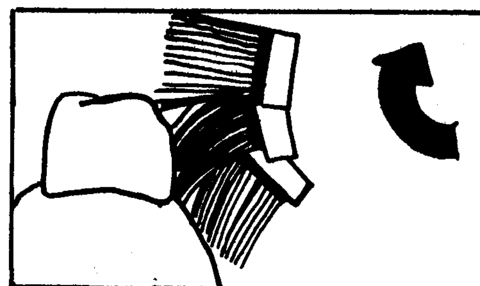
```
 154203                            (ENTRY KWOTE SUBR 1)
 154203 311 1  0 0  1232           (CAML 1 :BLIST)
 154204 263 17 0 0  0              (POPJ P)
 154205                            G101
 154205 514 1  0 0  1              (HRLZ 1 1)
 154206 250 1  0 16 2547           (EXCH 1 :MEM FREE)
 154207 250 16 0 0  1              (EXCH FREE 1)
 154210 505 1  0 0  22             (HRLI 1 'QUOTE)
 154211 250 1  0 16 2547           (EXCH 1 :MEM FREE)
 154212 250 16 0 0  1              (EXCH FREE 1)
 154213 263 17 0 0  0              (POPJ P)
 154214

 (154203 KWOTE SUBR 1)
                                   (END)
```

```
; 2 FACT------------------------------------------------------

(DE FACT (N) (IF (ZEROP N) 1 (TIMES N (FACT (SUB1 N)))))

 FUNCTION LENGTH = 19
 #LAP LENGTH = 11
;
;;;;;;
          (ENTRY FACT SUBR 1)
          (CAIN 1 '0)
          (JRST 0 :CRAONE)
 G103
          (PUSH P 1)
          (MOVE 5 :MEM 1)
          (SUBI 5 1)
          (JSP L :$CRANB)
          (PUSHJ P FACT)
          (POP P 2)
          (MOVE 5 :MEM 1)
          (IMUL 5 :MEM 2)
          (JRST 0 :CRANUM)
```



```
 (END)

 154214                          (ENTRY FACT SUBR 1)
 154214 306 1  0 0  13760        (CAIN 1 '0)
 154215 254 0  0 0  403145       (JRST 0 :CRAONE)
 154216                          G103
 154216 261 17 0 0  1            (PUSH P 1)
 154217 200 5  0 1  2547         (MOVE 5 :MEM 1)
 154220 275 5  0 0  1            (SUBI 5 1)
 154221 265 13 0 0  403152       (JSP L :$CRANB)
 154222 260 17 0 0  154214       (PUSHJ P FACT)
 154223 262 17 0 0  2            (POP P 2)
 154224 200 5  0 1  2547         (MOVE 5 :MEM 1)
 154225 220 5  0 2  2547         (IMUL 5 :MEM 2)
 154226 254 0  0 0  403126       (JRST 0 :CRANUM)
 154227

 (154214 FACT SUBR 1)
                                 (END)

; 3 ID1-----------------------------------------------------

(DE ID1 (L) (IF (NULL L) NIL L))

 FUNCTION LENGTH = 12
 #LAP LENGTH = 1
;
;;;;;;
        (ENTRY ID1 SUBR 1)
        (POPJ P)

 (END)

 154227                          (ENTRY ID1 SUBR 1)
 154227 263 17 0 0  0            (POPJ P)
 154230

 (154227 ID1 SUBR 1)
                                 (END)

; 4 FOO1----------------------------------------------------

(DE FOO1 (X) ['SETQ X ['CDR X]])

 FUNCTION LENGTH = 17
 #LAP LENGTH = 17
;
;;;;;;
        (ENTRY FOO1 SUBR 1)
        (HRLZI 2 0 1)
        (EXCH 2 :MEM FREE)
        (EXCH FREE 2)
        (HRLI 2 'CDR)
        (EXCH 2 :MEM FREE)
        (EXCH FREE 2)
        (HRLZ 2 2)
        (EXCH 2 :MEM FREE)
        (EXCH FREE 2)
        (HRL 2 1)
        (EXCH 2 :MEM FREE)
        (EXCH FREE 2)
        (MOVEI 1 0 2)
        (HRLI 1 'SETQ)
```

```
        (EXCH 1 :MEM FREE)
        (EXCH FREE 1)
        (POPJ P)
 (END)

 154232                          (ENTRY F001 SUBR 1)
 154232 515 2  0 1  0            (HRLZI 2 0 1)
 154233 250 2  0 16 2547         (EXCH 2 :MEM FREE)
 154234 250 16 0 0  2            (EXCH FREE 2)
 154235 505 2  0 0  1452         (HRLI 2 'CDR)
 154236 250 2  0 16 2547         (EXCH 2 :MEM FREE)
 154237 250 16 0 0  2            (EXCH FREE 2)
 154240 514 2  0 0  2            (HRLZ 2 2)
 154241 250 2  0 16 2547         (EXCH 2 :MEM FREE)
 154242 250 16 0 0  2            (EXCH FREE 2)
 154243 504 2  0 0  1            (HRL 2 1)
 154244 250 2  0 16 2547         (EXCH 2 :MEM FREE)
 154245 250 16 0 0  2            (EXCH FREE 2)
 154246 201 1  0 2  0            (MOVEI 1 0 2)
 154247 505 1  0 0  2046         (HRLI 1 'SETQ)
 154250 250 1  0 16 2547         (EXCH 1 :MEM FREE)
 154251 250 16 0 0  1            (EXCH FREE 1)
 154252 263 17 0 0  0            (POPJ P)
 154253

 (154230 F001 SUBR 1)
                                 (END)

; 5 XGQP-----------------------------------------------------

(DE XGQP (X Y Z)
    (REPEAT (ABS (DIFFER (SUB1 X) Y)) (PRIN1 Z))
    (BAR X Y))

 FUNCTION LENGTH = 24
 #LAP LENGTH = 18
;
;;;;;;
        (ENTRY XGQP SUBR 3)
        (JSP L :SBIND3)
        (XWD 'XGQP '(X Y Z))
        (GETVAL 1 X)
        (MOVE 5 :MEM 1)
        (SUBI 5 1)
        (GETVAL 2 Y)
        (SUB 5 :MEM 2)
        (MOVM 5 5)
        (MOVN 5 5)
        (PUSH P 5)
        (JRST 0 G108)
 G107
        (GETVAL 1 Z)
        (PUSHJ P :$PRIN1)
 G108
        (AOSG 0 0 P)
        (JRST 0 G107)
        (SUB P %T1)          ; (XWD 1 1) ;
        (MOVEI 1 '(BAR X Y))
        (JRST 0 EVAL)

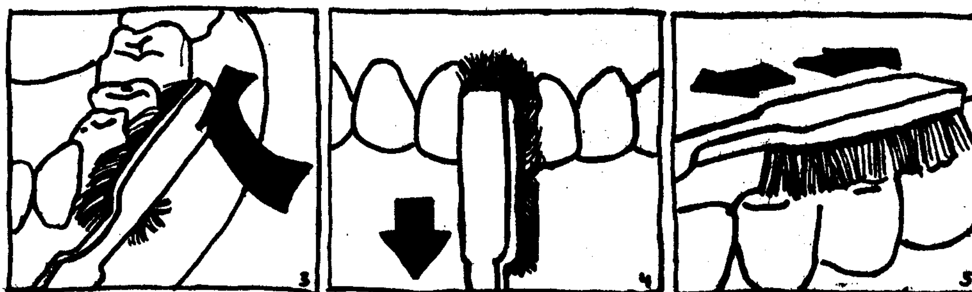
;---------- # T B L
 #TBL LENGTH = 1 ;
```

```
%T1 (XWD 1 1)

(END)

154253                              (ENTRY XGQP SUBR 3)
154253 265 13 0 0   414760          (JSP L :SBIND3)
154254      6030    61074           (XWD 'XGQP '(X Y Z))
154255 554 15 0 0   15265           (HLRZ 1 (:MEM 'X))
154256 200 5  0 1   2547            (MOVE 5 :MEM 1)
154257 275 5  0 0   1               (SUBI 5 1)
154260 554 20 0 0   16027           (HLRZ 2 (:MEM 'Y))
154261 274 5  0 2   2547            (SUB 5 :MEM 2)
154262 214 5  0 0   5               (MOVM 5 5)
154263 210 5  0 0   5               (MOVN 5 5)
154264 261 17 0 0   5               (PUSH P 5)
154265 254 0  0 0   0               (JRST 0 G108)
154266                              G107
154266 554 16 0 0   11751           (HLRZ 1 (:MEM 'Z))
154267 260 17 0 0   404116          (PUSHJ P :$PRIN1)
154270                              G108
154270 357 0  0 17  0               (AOSG 0 0 P)
154271 254 0  0 0   154266          (JRST 0 G107)
154272 274 17 0 0   0               (SUB P %T1)
154273 201 1  0 0   60721           (MOVEI 1 '(BAR X Y))
154274 254 0  0 0   405251          (JRST 0 EVAL)
154275                              %T1
154275      1       1               (XWD 1 1)
154276

(154253 XGQP SUBR 3)
                                    (END)
```



APPENDICE A

```
;           C O N F I G   . I N I                ;
;                                                ;
;------------------------------------------------;
;  CONFIGURATION standard pour VLISP 10 . 3      ;
; qui demande en mode conversationnel le type    ;
; de la configuration souhaitee.                 ;
;------------------------------------------------;

(IF
  (MEMQ
     (PROGN
        (STATUS 2 27)
        (WHILE (TYS) (TYI))
        (PRIN1 "Big config ?")
        (TERPRI -1)
        (ASCII (TYI)))
     (MAKLIST "OoYyTt"))  ; pour Oui ou Yes ou Tak ;

     (CONFIGURATION

            ; grosse config pour faire tout marcher ;

            '(DSK (VLISP . SYS))     ; fichier initial ;
            '(TTY (LISPIN . VLI))    ; fichier d'entrée ;
            '(TTY (LISPOU . LST))    ; fichier de sortie ;
            1000                     ; atomes totaux ;
            8000                     ; nbs standards ;
            400                      ; strings stds  ;
            25000                    ; listes ;
            1000                     ; pile syst stds ;
            5000                     ; pile user + arrays ;
            2000                     ; code ;
        )

      (CONFIGURATION

            ; petite config pour les tests simples ;

            '(DSK (VLISP . SYS))     ; fichier initial ;
            '(TTY (LISPIN . VLI))    ; fichier d'entrée ;
            '(TTY (LISPOU . LST))    ; fichier de sortie ;
            800                      ; atomes totaux ;
            500                      ; nbs standards ;
            100                      ; strings stds  ;
            8000                     ; listes ;
            1000                     ; pile syst stds ;
            600                      ; pile user + arrays ;
            200                      ; code ;
        )))))))
```

```
;          S Y S : V L I S P   .   I N I                    ;
;                                                           ;
;       Fichier initial standard de VLISP 10 . 3            ;
;-----------------------------------------------------------;
;                   Jérôme CHAILLOUX                        ;
;                                                           ;
;       Département d'Informatique                          ;
;       Université de Paris VIII - Vincennes                ;
;       Route de la Tourelle 75571 Paris Cedex 12           ;
;       Tél : 374 12 50 poste 299                           ;
;                                                           ;
;       I.R.C.A.M.                                          ;
;       31 Rue St Merri 75004 Paris                         ;
;       Tel : 277 12 33 poste 48-48                         ;
;-----------------------------------------------------------;

(STATUS 2 0 1 2) ; silence !!! ;

;
```

```
;
; ERROR.UBV  ERROR.UDFE  ERROR.UDFA  et  ESCAPE.I ;

; Définition des traps erreurs ;

(DE ERROR.UBV (atome pile p$bind)
    (PRINT "Variable indefinie : " atome)
    (OR (EQ p$bind -1)  ; s'il y a des "eval-frames" dans la pile ;
        (PROGN
            (PRINTLEVEL 6)
            (PRINTLENGTH 10)
            (PRINT "La derniere FONCTION etait : " (LASTCALL 2))
            (PRINTLEVEL 50)
            (PRINTLENGTH 2000)))
    (OUTPUT)
    (RESET))


(DE ERROR.UDFE (fonction forme pile p$bind)
    ; Undefined function EVAL ;
    (ERROR.UDF "Fonction indefinie dans EVAL : "))


(DE ERROR.UDFA (fonction forme pile p$bind)
    ; Undefined function APPLY ;
    (ERROR.UDF "Fonction indefinie dans APPLY : "))


(DE ERROR.UDF (msg)
    ; Fonction générale d'erreur FUNCTION UNDEFINED ;
    (PRINT msg fonction)
    (PRINTLEVEL 6)
    (PRINTLENGTH 10)
    (PRINT "La derniere forme etait : " forme)
    (OR (EQ p$bind -1)
            (PRINT "La derniere FONCTION etait : " (LASTCALL 3)))
    (PRINTLEVEL 50)
    (PRINTLENGTH 2000)
    (OUTPUT)
    (RESET))


(DE ESCAPE.I (numero pile p$bind lu IT)
                        ; dans ce traitement le numéro n'est pas utilisé ;
   (TERPRI)                              ; impression de l'avertissement ;
   (PRINT "Je rentre dans un TOPLEVEL ESCAPE-I.")
   (PRINT "Pour en sortir, commence une ligne par <META-ESPACE>.")
   (STATUS 11 '/!)                       ; changement du prompt top-level ;
   (TEREAD)                                    ; vide le buffer d'entrée ;
   (UNTIL (EQ (PEEKCH) '/ )
      (SETQ lu (READ))
      (SETQ IT (PRINT (EVAL lu))))        ; pour conserver le IT feature ;
   (STATUS 11 '/?)                        ; restaure le prompt top-level ;
   (PRINT "Ca roule..."))                  ; vers de nouvelles aventures ;
;
```

```
;
;;; FILESPEC : spécification d'un fichier ;;;

(DE FILESPEC (oldspec ;; spec tok next lnext dev filename ext prj prg pro)
   ; traite une spécification de fichier générale. ;
   ; si <spec> est une liste ou un atome ne fait rien mais ;
   ; si <spec> est une chaîne, ramène la forme normale VLISP ;
   ; ex : "dev:filname.ext[prj.prg]<pro>" est traduit en ;
   ;        (dev (filename . ext) (prj . prg) pro)   ;
   (SETQ spec oldspec)
   (IF (LISTP spec) (LESCAPE spec))
   (IF (LITATOM spec) (LESCAPE spec))
   (IF (NUMBP spec) (LESCAPE (FILESPECERROR)))
   ; la spécification est donc une chaîne ;
   (STATUS 5 8) ; pour les conversions octales ;
   (SETQ spec (MAKLIST spec)
         lnext '((/: . filename) (/. . ext) (/[ . prj)
                 (/, . prg) (/] . pro) (/< . pro))
         next 'filename)
   (WHILE spec
      (SETQ tok ())
      (WHILE (AND spec (NOT (MEMQ (CAR spec) '(/: /. /[ /, /] /< />))))
         (SETQ tok (CONS (NEXTL spec) tok)))
      (SETQ next
           (OR  (IF (NULL spec) next)
                (CASSQ (CAR spec) '(( /: . dev) (/. . filename) (/< . pro)
                       (/[ . ext) (/, . prj) (/] . prg) (/> . pro)))))
      (IF (CAR next) (LESCAPE (FILESPECERROR))
           (SET next (IMPLODE (REVERSE  tok))))
      (SETQ next (CASSQ (CAR spec) lnext))
      (IF (SETQ tok (CASSQ (CADR spec) lnext))
          (SETQ spec (CDR spec)
                next tok))
      (SETQ spec (CDR spec)))
   (STATUS 5 10) ; repasse en mode décimal en entrée ;
   (COND
      (pro [dev [filename . ext] [prj . prg] pro])
      ((OR prj prg) [dev [filename . ext] [prj . prg]])
      (T [dev [filename . ext]])))


(DE FILESPECERROR ()
   ; ya eu une erreur, on redemande le nom du fichier ;
   (PRINT "FILESPEC specification incorrecte dans :" oldspec)
   (PRIN1 "Tape la nouvelle specification") (TERPRI -1)
   (FILESPEC (READSTR)))
;
```

```
;
;;; FILE : Acces au systeme de gestion de fichiers ;;;

(DF FILE (cmd ;; file status)
   ;;
   ; appel (FILE <operation> <1er fichier> <2eme fichier>) ;
   ; avec <operation> non-evalue ;
   ;;
   (SETQ file (FILESPEC (EVAL (CADR cmd))))
   (SETQ status

      (SELECTQ (CAR cmd)
          ;;
          ;  différentes <opérations> ;
          ;;

        (READ
           ; sélection du fichier d'entrée ;
           ; syntaxe : (FILE READ <file> NIL/T)  ;
           ; équivalent à (INPUT file) si l'indic = NIL ;
           ; ne traite pas les directory à la ETV si indic = T ;
           (OR (EQ (CAR file) 'TTY) (STATUS 2 11))
           (OR (NUMBP (FILOP [1 1] file))
               (IF (AND (NOT (CADDR cmd))    ; teste de l'indicateur ;
                        (EQ (PEEKCH) 'C)) ; saute le directory de l'IRCAM ;
                   (UNTIL (EQ (READCH) '/;)))))

        (CREATE
           ; sélection du fichier de sortie ;
           ; si le fichier existait déjà, ça ;
           ; provoque une erreur ;
           (FILOP [2 2] file))

        (WRITE
           ; ouvre le fichier file en sortie ;
           ; équivalent à la fonction OUTPUT ;
           (FILOP [2 3] file))

        (APPEND
           ; ouvre le fichier file en sortie ;
           ; si le fichier existait déjà, les sorties ;
           ; se feront en FIN de celui-ci ;
           (FILOP [2 6] file))

        (CLOSE
           ; ferme un fichier. syntaxe spéciale ;
           ; (FILE CLOSE () <n>) ;
           ; avec <n>=1 pour le fichier d'entrée  ;
           ; avec <n>=2 pour le fichier de sortie ;
           (FILOP [(CADDR cmd) 7]))

        (CHECK
           ; chekpoint le fichier de sortie ;
           ; i.e. préserve l'état courant de ce fichier ;
           ; équivalent à  (FILOP CLOSE () 2) ;
           ; suivi de (FILOP APPEND le-même-fichier-de-sortie) ;
           (FILOP [2 \10]))

        (RENAME
           ; change le nom du fichier file en file2 ;
           (FILOP [3 \13] file (FILESPEC (EVAL (CADDR cmd))))
```

```
        (DELETE
           ; détruit le fichier de nom file ;
           (FILOP [3 \14] file))

        (-1)))

   (IF (NUMBP status)
        (PROGN
          (OUTSTR (CONCAT "FILE : " (CAR cmd) " "
           ((+ status 2) '(
                "Fonction inconnue"
                "Fichier inexistant"
                "Directory inexistant"
                "Protection trop forte"
                "Fichier en cours de modification"
                "Fichier existant"
                "Sequence illegale"))))
           (TERPRI)
           (RESET))
        file        ; ramene file ;  )))))
;
```

```
;
;;; MACROS STANDARD ;;;

; Pour des liaisons dynamiques de variables ;

(DM LET (ls) (RPLACB ls
  (CONS (MCONS LAMBDA (MAPCAR (CADR ls) 'CAR) (CDDR ls))
        (MAPCAR (CADR ls) 'CADR)))))

; Pour des liaisons dynamiques de fonctions ;

(DM WITH (l)
   (RPLACB l
      ['PROGN
       ['ADDPROP [QUOTE (CAADR l)] [LAMBDA . (CDADR l)] EXPR]
       ['PROG1 ['PROGN . (CDDR l)]
               ['REMPROP [QUOTE (CAADR l)] EXPR]]]))

; les ESCAPEs INFINIS :                        ;
; (ESCLOOP <at> <s1> ... <sN>)  ==             ;
; (ESCAPE <at> (WHILE T <s1> ... <sN>))        ;

(DM ESCLOOP (l)
   (RPLACB l
      ['ESCAPE (CADR l)
         ['WHILE T . (CDDR L)]]))

; et les REPEAT UNTIL ;

(DM REPEATUNTIL (l)
   (RPLACB l
      ['WHILE ['PROGN . (CDR l)]])))

; (BACKTRACK nom (lvar) essai1 essai2 ... essaiN ) ;

(DM BACKTRACK (call ;; nom-echec lvars lvals lclauses
                       lnewvars lrestore)
  (SETQ nom-echec (CADR call)
        lvars     (MAPCAR (CADDR call) (LAMBDA (X) (IF (ATOM X) X (CADR X))))
        lvals     (CADDR call)
        lclauses  (CDDDR call)
        lnewvars  (MAPCAR lvars (LAMBDA (X) (GENSYM 'SAV// X)))
        lrestore  (CONS 'SETQ
                     (LET ((X lvars) (Y lnewvars))
                        (IF X (MCONS (NEXTL X) (NEXTL Y) (SELF X Y))))))
  (RPLACB call
    ['ESCAPE '$ESSAI$
      (CONS (MCONS LAMBDA lnewvars
               (NCONC1 (LET ((X lclauses))
                          (IF X (MCONS ['ESCAPE nom-echec
                                                  ['$ESSAI$ (NEXTL X)]]
                                       lrestore
                                       (SELF X))))
                       ''ECHEC))
            lvals)]))
;
```

```
;
;;; SYNONYM ;;;

(SYNONYM 'GTZ 'GZP)        ; pour rendre Harald HEUREUX ;
(SYNONYM '=0 'ZEROP)       ; tout ça pour faire comme en SAIL ! ;
(SYNONYM '#0 'NEROP)
(SYNONYM '>0 'GZP)
(SYNONYM '<0 'LZP)
(SYNONYM '>=0 'GEZP)
(SYNONYM '<=0 'LEZP)

;;; Macro-caractères ;;;

(DMC /λ () ;↑H; 'LAMBDA)
(DMC /r () ;↑Q; (STATUS 1 5) (STATUS 21) (STATUS 2 5) NIL)
(DMC /â () ;↑A; ['LIBRARY (READ)])
(DMC /← () ;  ; (STOP))
(DMC /⊏ () ;↑P; ['PRETTY (READ)])
(DMC /è () ;↑F; ['PHENARETE (READ)])
(DMC /+ () ;↑G; (DISPLAY '(\177 7)) '/+)

;;; Utilise le nouveau trait RUN ;;;

(DMC /┘ () ;↑W; (RUN '(SYS (WHO . SAV))))
(DMC /é () ;↑E; (RUN '(SYS (E . SHR)) -1))

;;; Fonctions spéciales sur DATA-MEDIAS ;;;

(DE TTYDMP ()
   ; teste si le terminal utilisé est en TTY DM mode ;
   ; ramène NIL si faux (sinon ramène un nb qcq) ;
   (LZP (TRMOP \1043 () ())))

(DE ESCAPECH (c n)
   ; simule l'envoi d'un caractère ESCAPE n c ;
   (TRMOP \33 0 (PLUS N (SWAP (CASCII c)))))

(DE BREAKCH (c n)
   ; simule l'envoi d'un caractère BREAK n c ;
   (TRMOP \33 0 (PLUS N (SWAP (LOGOR \400000 (CASCII c))))))

(DE WHOLINE (n)
   ; affiche la WHO line du job n ;
   (ESCAPECH 'W n))
;
```

```
;
;;; Autres fonctions sur fichiers disque ;;;

(DF DUMPF (ls ;; filout)
    ; (DUMPF <file> <fn1> ... <fnN>) ;
    ; cré un fichier <file> contenant les définitions ;
    ; des fonctions <fn1> ... <fnN> ;
    (SETQ filout ['DSK [(NEXTL ls) . 'VLI] () \055])
    (OUTPUT filout)
    (WHILE ls (EVAL ['PRETTY (NEXTL ls)]))
    (OUTPUT)
    filout)

(DE TYPE (filin)
   ; simule la commande moniteur .TYPE file ;
   (FILE READ filin)
   (STATUS 17 (ASCII \15) 2)
   (WITH (EOF () (TERPRI) (INPUT) (&eof))
      (ESCLOOP &eof (PRINC (READCH))))
   (STATUS 17 (ASCII \15) 0)
   filin)

(DF HELP (file)
   ; simule la commande moniteur :  .HELP file ou .HELP VLISP ;
   (TYPE ['HLP [(OR (CAR file) 'VLISP) . 'HLP]]))


; Ces fonctions permettent de connaitre le nom et ;
; l'identification des gens qui travaillent sur le PDP 10 ;

(DF whois (name ;; ligne jelai)
     ; (WHOIS nom) ramene le nom du gars ;
     ; ca fait ca intelligement :  ;
     ; (WHOIS JEROME) ->  "JER Jerome Chailloux"    ;
     ; (WHOIS JER)    ->  "JER Jerome Chailloux"    ;
     ; (WHOIS CHAILLOUX) ->  "JER Jerome Chailloux" ;
     ; (WHOIS JE) -> tous les noms qui commencent par JE ;
     (SETQ name (CAR name))
     (WITH (EOF () (INPUT) (&eof))
        (FILE READ "SYS:FACT.TXT" T)
        (ESCLOOP &eof
          (SETQ ligne (READSTR))
             (ESCAPE &uncoup
                (MAPC (IMPLODE (CONCAT "(" ligne ")"))
                   (LAMBDA (nom)
                     (IF (SAMEPN nom name)
                       (&uncoup (PRINT ligne) (SETQ jelai T))))))))
     (IF jelai  'OK "Nie ma ..."))))

(DE whoisall ()
      ; liste tous les utilisateurs potentiels ;
      ; appel : (WHOISALL) c'est tout ;
      (WITH (EOF () (INPUT) (&eof))
         (FILE READ "SYS:FACT.TXT" T)
         (ESCLOOP &eof (PRINT (READSTR))))
      'OK)
;
```

```
;
; Fonctions autoloadables ;

(PATHLIBRARY ()
        ; directories utiles ;
         SYS
        (LIS . JER)
        (LIS . PAT)
        (LIS . HAR)
        (LIS . GOO)
        (LIS . LOU))


(AUTOLOAD AID
        PACKFILE
        SIZE
        SIZEFILE)


(AUTOLOAD DISPLAY
        DPYINIT
        DPYEND
        DPYS
        DPYRECHO
        DPYRNB
        DPYRSTR
        DPYCURSOR)

;
(if (eq 5 (REM (QUO (STATUS 36) 1000) 10))
(repeat 50
(print "Non, rien de rien, non, je ne regrette rien")))
;

(DE OUTSTR (str)
   ; equivalent de l'UUO OUTSTR ; ;
   ; i.e. ecrit sur le terminal la chaine <str> ;
   (MAPC (MAPCAR (MAKLIST str) 'CASCII) 'TYO)
   str))
;
```

```
;
; final : IDENTIFICATION et lecture DSK:VLISP.INI ;

(PROGN
        ; init de la taille des ecrans DATA-MEDIAS ;
        (AND (IRCAMP) (STATUS 9 76))
        ; edition du numero de version, date, heure et PPN ;
        (SETQ VERSION (VERSION))
        (PRIN1
           (SETQ VERSION
              (GENSYM
                'VLISP
                '/
                (LOGAND \777 (LOGSHIFT (SWAP VERSION) -6))
                '/.
                (LOGAND \77 (SWAP VERSION))
                (MINUS (LOGAND \777777 VERSION))))
           (DATE)
           (TIME)
           (GETPPN))
        (STATUS 1 20) (TERPRI)
        (STATUS 2 27)
        (PRINT "Pour avoir l'etat de la documentation demande .HELP VLISP ")
        (STATUS 1 27)
        (OUTSTR "SYS:VLISP.INI loaded.")
        (TERPRI)
        (WHILE (TYS) (TYI))
        (IFN (DIRECTORY () '(VLISP . INI))
           ; le fichier DSK:VLISP.INI n'existe pas : passage en mode TTY
           (PROGN (INPUT) (STATUS 1 0 1) 'PRET)
           ; sinon le fichier DSK:VLISP.INI existe ;
           (FILE READ "DSK:VLISP.INI")
           (STATUS 2 0 1 2) ; la lecture est silencieuse ;
           (DE EOF () (INPUT) (STATUS 1 0 1) 'PRET))))))
```

```
;     Fichier initial VLISP 10 . 3     (LIS . JER)       ;
;                                                         ;
;---------------------------------------------------------;
;    Permet le chargement du dernier fichier édité        ;
;    apres impression d'une demande sur TTY               ;
;---------------------------------------------------------;

(STATUS 2 0 1 2)

; silence !!! ;

; READFTMPCOR : lecture du TMPCOR de ETV ;

(DE READFTMPCOR ( ;; l filin)
   (SETQ l (TMPCOR 'ED)) ; pour TECO a l'IRCAM 'EDS ;
   (IFN (LISTP l) (LESCAPE))
   (OR
      (AND (EQ (NEXTL l) 'E) (EQ (NEXTL l) 'T) (NEXTL l))
      (LESCAPE))
   (SETQ filin)
   (WHILE (AND l (NOT (MEMQ (CAR l) '(/. /( ))))
      (SETQ filin [(NEXTL l) . filin]))
   (NEXTL l)
   (OR
      (AND
         (EQ (NEXTL l) 'V)
         (EQ (NEXTL l) 'L)
         (EQ (NEXTL l) 'I))
      (LESCAPE))
   (SETQ filin (APPLY 'GENSYM (REVERSE filin))))

; lecture ou non du dernier fichier edite ;

(IF
   (SETQ filin (READFTMPCOR))
   (PROGN
      (IFN
         (OR
            (STATUS 4 29)   ; J'arrive de ETV  ou ;
            (MEMQ
               (PROGN
                  (WHILE (TYS) (TYI))
                  (STATUS 2 27)
                  (PRIN1 (CONCAT "On charge " filin ".VLI ?"))
                  (TERPRI -1)
                  (ASCII (TYI)))
               (MAKLIST "OoYyTt")))
         (PROGN
            (REMPROP 'EOF EXPR)
            (WHILE (TYS) (TYI))
            (INPUT)
            (STATUS 1 0 1)
            'PRET)
         (INPUT filin)
         (STATUS 2 0 1 2)
         (DE EOF ()
            (REMPROP 'EOF EXPR)
            (OUTSTR (CONCAT "DSK:" filin ".VLI loaded."))
            (TERPRI)
            (WHILE (TYS) (TYI))
            (INPUT)
            (STATUS 1 0 1)
            'PRET))))
```

```
(POUR EVAL (OUTSTR "DSK:VLISP.INI loaded."))
```

APPENDICE B

```
;            D I S P L A Y   .   V L I                         ;
;                                                              ;
;      Fonctions de visualisation sur les terminaux            ;
;            DATAMEDIA en VLISP 10 . 3                         ;
;--------------------------------------------------------------;
;                    Jerome CHAILLOUX                          ;
;                                                              ;
;      Departement d'Informatique                              ;
;      Universite de Paris VIII - Vincennes                    ;
;      Route de la Tourelle 75571 Paris Cedex 12               ;
;      Tel : 374 12 50 poste 299                               ;
;                                                              ;
;      I.R.C.A.M.                                              ;
;      31 Rue St Merri 75004 Paris                             ;
;      Tel : 277 12 33 poste 48-48                             ;
;--------------------------------------------------------------;
;                                                              ;
; Fonctions de type DPYxxx                                     ;
; Ces fonctions permettent d'avoir un controle complet         ;
; sur tout l'ecran en particulier le "line-editor" standard;
; ainsi que le READ  VLISP ne sont plus actifs                 ;
;                                                              ;
; Les seules fonctions standard disponibles sont :             ;
;      TYI     qui lit 1 caract sur le clavier SANS echo       ;
;      DISPLAY qui envoie une suite de code internes           ;
;              a l'ecran                                       ;
;      PPIOT   qui effectue des fonctions speciales            ;
;                                                              ;
; Toutes ces fonctions sont AUTOLOAD sur SYS:DISPLAY.VLI       ;
;--------------------------------------------------------------;
;;
; pour etre lu de tous temps ;

(PROGN
   (SETQ READ.TABLE  (READ.TABLE))
   (READ.STD)
   'PRET)
```

```
; DPYINIT DPYEND DPYERROR DPYS ;

(DE DPYINIT ()
  ; passe en mode DPYxxx, i.e. efface l'ecran ... ;
  ; pour en sortir faire (DPYEND) qui restaure tout ;
  (STATUS 2 6)  ; enleve le wholine GC ;
  (PPIOT 0 2)   ; passage sur la page 2 ;
  (CALLI \-10 1) ; FREEZE 1, ;
  (DISPLAY '(\177 \36)) ; envoie du master clear ;
  'DPYINIT)

(DE DPYEND ()
  ; repasse en mode normal ;
  ; i.e. (PROGN (DPYINIT) (DPYEND)) ne fait rien ;
  (CALLI \-10 0) ; FREEZE 0, ;
  (CALLI -7) ; DPYCLR ;
  (STATUS 1 6) ;remet le wholine GC ;
  (TRMOP \33 0 \-377662000000) ; equivalent de [NUL]-N ;
  (PRINT 'DPYEND))

(DE DPYERROR (~msg)
   ; erreur dans une fonction DPYxxx ;
   ; reinit tout et edite le message ~msg ;
   (DPYEND)
   (PRINT 'DPYERROR ': ~msg)
   ())

(DE DPYS (~xcursor ~ycursor ~msg)
   ; edite la chaine ~msg sur un ecran en DPYxxx mode ;
   ; en position : ~ycursor ligne, ~xcursor colonne ;
   (DISPLAY (APPEND [\177 \14 (LOGXOR \140 ~xcursor)
                              (LOGXOR \140 ~ycursor)]
                  (MAPCAR (MAKLIST ~msg) 'CASCII)))  )))
```

```
; DPYRECHO DPYRSTR DPYCURSOR DPYRNB ;

(DE DPYRECHO ( ;; ~char)
   ; lit un caractere et l'echo ;
   ; suppose ~xcursor et ~ycursor globaux et contenant ;
   ; la position courante du curseur ;
   ; ne traite que la visualisation  du DEL ;
   (SETQ ~char (TYI))
   (IF (EQ ~char \177)
      ; c'est pas un DEL ;
      (DPYS (INCR ~xcursor) ~ycursor (STRING (ASCII ~char)))
      ; c'est un DEL ;
      (DPYS ~xcursor ~ycursor " ")
      (DECR ~xcursor))
   ~char)

(DE DPYRSTR (~xcursor ~ycursor ~msg ;; ~str ~char)
   ; edite le message ~msg a la position ~xcursor, ~ycursor
   ; puis lit une chaine de caracteres terminee par 'Return' ;
   (DPYS ~xcursor ~ycursor (SETQ ~msg (STRING ~msg)))
   (SETQ ~xcursor (PLUS ~xcursor (STRINGL ~msg)))
   (UNTIL (EQ (SETQ ~char (TYI)) \15)
      (IF (NEQ ~char \177)
         ; c'est pas un DEL ;
         (PROGN
            (DPYS (INCR ~xcursor) ~ycursor (STRING (ASCII ~char)))
            (NEWL ~str (ASCII ~char)))
         ; c'est un DEL ;
         (DPYS ~xcursor ~ycursor " ")
         (DECR ~xcursor)
         (NEXTL ~str)))
   (STRING (REVERSE ~str)))

(DE DPYCURSOR (~xcursor ~ycursor)
   ; positionne le curseur en XCURSOR et YCURSOR ;
   (OR (PPIOT 8 (LOC (LOGOR (LOGSHIFT ~xcursor 18) ~ycursor) ()))
      (DPYERROR ['DPYCURSOR ~xcursor ~ycursor])))

(DE DPYRNB (~xcursor ~ycursor ~msg ;; ~str ~n)
   ; imprime le message ~msg en ~xcursor et ~ycursor ;
   ; et lit a la suite (sur la meme ligne) un nombre decimal ;
   ; la conversion s'effectue par la fonction elle-meme ;
   ; ca ramene le nb convertit lu ;
   (SETQ ~str (MAKLIST (DPYRSTR ~xcursor ~ycursor ~msg)))
   (IF (NULL ~str) (LESCAPE 0))
   (SETQ ~xcursor (PLUS ~xcursor (STRINGL ~msg)))
   ; saute tous les caracteres non significatifs ;
   (WHILE (NOT (NUMBP (SETQ ~n (NEXTL ~str))))
        (IF (NULL ~str) (LESCAPE 0)))
   ; effectue vraiment la lecture ;
   (WHILE (NUMBP (CAR ~str))
          (SETQ ~n (PLUS (TIMES ~n 10) (NEXTL ~str))))
   ; ramene le nb convertit ;
   ~n))

; epilogue standard ;

(READ.TABLE READ.TABLE)
```

```
;                HANOI . VLI                                 .;
;                                                             ;
;          Les Tours de HANOI      VLISP 10 . 3               ;
;       Test de l'utilisation des ecrans DATA-MEDIAS          ;
;-------------------------------------------------------------;
;                Jerome CHAILLOUX                             ;
;                                                             ;
;       Departement d'Informatique                            ;
;       Universite de Paris VIII - Vincennes                  ;
;       Route de la Tourelle 75571 Paris Cedex 12             ;
;       Tel : 374 12 50 poste 299                             ;
;                                                             ;
;       I.R.C.A.M.                                            ;
;       31 Rue St Merri 75004 Paris                           ;
;       Tel : 277 12 33 poste 48-48                           ;
;-------------------------------------------------------------;

; Cette fonction est d'habitude sur VLISP.INI ;

(DE TYPE (filin)
   ; simule la commande moniteur .TYPE file ;
   (FILE READ filin)
   (STATUS 17 (ASCII \15) 2)
   (WITH (EOF () (TERPRI) (INPUT) (&eof))
      (ESCLOOP &eof (PRINC (READCH))))
   (STATUS 17 (ASCII \15) 0)
   filin)


(DE PAIG (L XPOS ;; y)
   ; imprime le contenu de toute une aiguille ;
   ; cette fonction sert a initialiser tout le monde ;
   (SETQ y ypos)
   ; affiche la parti nue de l'aiguille ;
   (REPEAT (DIFFER maxdsk (LENGTH L))
      (DPYS (DIFFER xpos maxdsk) y gom) (SETQ y (ADD1 y)))
   ; affiche les disque sur l'aiguille ;
   (WHILE (LISTP L)
      (DPYS (DIFFER XPOS (CAR L))
         (PLUS ypos (DIFFER maxdsk (LENGTH L)))
         (CONCAT (DUPL (CAR L) (CAR L)) "*" (DUPL (CAR L) (CAR L))))
      (NEXTL L))
   ; affiche la base de l'aiguille ;
   (DPYS (DIFFER XPOS maxdsk) (PLUS ypos maxdsk)
         (DUPL "-" (ADD1 (PLUS maxdsk maxdsk)))))

; le tableau d'occupation contient pour chaque aiguille ;
; le numero du disque ;
(DA 'TABOCC 4)

; le tableau des positions des aiguilles ;
; contient la position Y de chaqu aiguille ;
(DA 'XAIG 4 (LAMBDA (X) (X '(10 33 56))))
```

```
; UP GO DOWN VISUT HANOI.REC ;
; Fonctions de visualisation d'un mouvement ;

(DE UP (depart ;; l x)
   ; le disque se leve ;
   (SETQ
        l (DIFFER maxdsk (LENGTH (TABOCC depart)) )
        ; position / au debut de l'aiguille ;
        x (DIFFER (XAIG depart) n))
   (REPEAT l
        (SETQ l (SUB1 l))
        (DPYS x (PLUS ypos l) disk)
        (DPYS x (ADD1 (PLUS ypos l)) gomm)))))

(DE GO (depart arrivee ;; fnt x1 x2 x d)
   ; voyage au dessus des aiguilles ;
   (SETQ x1 (XAIG depart) x2 (XAIG arrivee) x (DIFFER x1 n))
   (SETQ fnt (IF (GT x1 x2) 'DECR 'INCR))
   (SETQ d (CONCAT " " disk " "))
   (DPYS (ADD1 (DIFFER x1 maxdsk)) ypos gom)
   (REPEAT (ABS (DIFFER x1 x2))
        (DPYS x ypos d)
        (fnt x))
   (DPYS (ADD1 (DIFFER x2 maxdsk)) ypos gom)))

(DE DOWN (arrivee ;; x y)
   ; le disque se rabaisse ;
   (SETQ
        x (DIFFER (XAIG arrivee) n)
        y ypos)
   (REPEAT (SUB1 (DIFFER maxdsk (LENGTH (TABOCC arrivee))  ))
        (DPYS x (ADD1 y) disk)
        (DPYS x y gomm)
        (SETQ y (ADD1 y)))))))

(DE VISUT (n depart arrivee ;; disk gomm)
   ; visualise le deplacement complet du disque n ;
   (DPYS 51 3 (SETQ nmouv (ADD1 nmouv)))
   (SETQ disk (CONCAT (DUPL n n) "*" (DUPL n n))
         gomm (CONCAT (DUPL " " n) "*" (DUPL " " n)))
   (UP depart)
   (GO depart arrivee)
   (DOWN arrivee)
   (SETQA TABOCC depart (CDR (TABOCC depart)))
   (SETQA TABOCC arrivee (CONS n (TABOCC arrivee))))


(DE HANOI.REC (n depart arrivee inter)
   ; fonction recursive de calcul des mouvements ;
   (IF (LEZP n) ()
        (HANOI.REC (SUB1 n) depart inter arrivee)
        (VISUT n depart arrivee)
        (HANOI.REC (SUB1 n) inter arrivee depart))))
```

```
; HANOI ;

(DE HANOI ( ;; maxdsk ndsk n nmouv)
   ; sequenceur principal ;
   ; lecture du baratin ;
   (IFN (DIRECTORY '(LIS . JER) '(HANOI . DOC))
        (PRINT "Y a pas de fichier DSK:HANOI.DOC[LIS,JER].")
        (TYPE '(DSK (HANOI . DOC) (LIS . JER)))
        (UNTIL (TYI)) ; attente de la frappe de l'espace ; )
   (DPYINIT) ; passage sur la page 1;
   (DPYS 25 3 "Les Tours de HANOI.")
   ; initialisation du nb maximum de disques ;
   (SETQ maxdsk 9)
   (WHILE T
      (DPYS 25 6 "(0 pour terminer HANOI.)")
      (DPYS 25 5 "Combien voulez-vous de disques ?")
      ; initialisation du nb courant de disques ;
      (WHILE (OR (LZP (SETQ ndsk (DIFFER (TYI) \60)))
                 (GT ndsk 9)))
      (OR (NEROP ndsk)
        (LESCAPE (DPYEND)
                ; pour faire peur aux petites filles ;
                (PRINT "Tape 'RETURN' pour provoquer la fin du monde.")
                (IF (NEQ (TYI) \15) (LESCAPE (INPUT) (RESET)))
                (PRINT "Deleted All Files")
                ; passe sur le ppn de l'utilisateur ;
                (ALIAS)
                (MAPC (DIRECTORY)
                   (LAMBDA (L) (PRINT (CAR L) '/. (CDR L) '/ / DELETED)))
                ; c'est vraiment la fin ;
                (RUN '(SYS(KJOB)))
          'Hanoi))
      ; effacement des messages ;
      (DPYS 51 3 "          ")
      (DPYS 25 5 "                                    ")
      (DPYS 25 6 "                                    ")
      ; 1ere ligne vide pour les aiguilles ;
      (SETQ ypos 12)
      ; calcul l'emplacement d'un disque vide ;
      (SETQ gom (CONCAT (DUPL " " maxdsk) "*" (DUPL " " maxdsk)))
      (SETQ n [ndsk])
      (REPEAT (SUB1 ndsk) (SETQ n (CONS (SUB1 (CAR n)) n)))
      (SETQA TABOCC 1 n) (SETQA TABOCC 2 ()) (SETQA TABOCC 3 ())
      (PAIG (TABOCC 1) (xaig 1))
      (PAIG (TABOCC 2) (xaig 2))
      (PAIG (TABOCC 3) (xaig 3))
      ; init du nb de position ;
      (SETQ nmouv 0)
      (HANOI.REC ndsk 1 2 3)))))


(POUR EVAL (PRINT "Pour lancer il faut taper : (HANOI) "))
```

APPENDICE C

```
!        T R A C E   /   S T E P                          !
!                                                         !
!        Trace et Pas-a-pas    VLISP 10 . 3               !
!---------------------------------------------------------!
!             Jerome CHAILLOUX                            !
!                                                         !
!        Departement d'Informatique                       !
!        Universite de Paris VIII - Vincennes             !
!        Route de la Tourelle 75571 Paris Cedex 12        !
!        Tel : 374 12 50 poste 299                        !
!                                                         !
!        I.R.C.A.M.                                       !
!        31 Rue St Merri 75004 Paris                      !
!        Tel : 277 12 33 poste 48-48                      !
!---------------------------------------------------------!

; Pour pouvoir lire ce fichier en tous temps ... ;

(PROGN (SETQ READ.TABLE (READ.TABLE)) (READ.STD) 'PRET)

; pour distinguer les PRINT ordinaires des PRINT generes par TRACE ;

(SYNONYM 'PRINTRACE 'PRINT)
```

```
; TRACE ;

(DF TRACE (%F ;; %X)
   ;;
   ; Force une TRACE de n'importe quelle fonction de type ;
   ; SUBR FSUBR EXPR FEXPR MACRO MACIN MACOUT ;
   ;;
   (OR
      (LISTP %F)
      ; y a rien a tracer ?!? ;
      (LESCAPE "** TRACE quoi ?"))
   ; TRACE contient la liste des fonctions (cf UNTRACE) ;
   ; pose des indicateurs ;
   (MAPC %F
      (LAMBDA (%A)
         (COND
            ((MEMQ (TYPEFN %A) '(SUBR FSUBR))
                (STATUS 28 %A))
            ((SETQ
                %X (GETL %A '(EXPR FEXPR MACRO MACIN MACOUT)))
               (PUT %A 0 'TRACE)
               (PUT %A
                  [LAMBDA
                   (CADR (CADR %X))
                   ['PUT
                     [QUOTE %A]
                     ['ADD1 ['GET [QUOTE %A] ''TRACE]]
                     ''TRACE]
                   ['PRINTRACE
                     ['GET [QUOTE %A] ''TRACE]
                     [QUOTE '--->]
                     [QUOTE %A]
                     [QUOTE ':]
                     (IF (ATOM (CADR (CADR %X)))
                         (CADR (CADR %X))
                         ['LIST . (CADR (CADR %X))]])]
                   ['PRINTRACE
                     ['GET [QUOTE %A] ''TRACE]
                     [QUOTE '<---]
                     [QUOTE %A]
                     [QUOTE '=]
                     ['PROG1
                         ['PROGN . (CDDR (CADR %X))]
                         ['PUT
                           [QUOTE %A]
                           ['SUB1 ['GET [QUOTE %A] ''TRACE]]
                           ''TRACE] ]]]
                  (CAR %X)))
            (T (SETQ %F (DELQ %F %A))
               (LESCAPE (PRINT "** TRACE : C'est quoi " %A))))
         ; ajoute dans la liste des fonctions tracees ;
         (IF (BOUNDP 'TRACE)
            (NEWL TRACE %A)
            (SETQ TRACE [%A]))))
   %F)
```

```
; TRACEFILE TRACEF UNTRACE ;

(DE TRACEFILE (file ;; l)
   ; lit et TRACE toutes les definitions du fichier file ;
   (INPUT file)
   (WITH (EOF () (INPUT) (&eof))
   (ESCLOOP &eof
      (SETQ l (READ))
      (EVAL l)
      (IF (AND (LISTP l)
               (MEMQ (CAR l) '(DE DF DM DMI DMO DEFUN)))
          (EVAL ['TRACE (CADR l)])))
   file))

(DF TRACEF (l)
   ; equivalent FSUBR de TRACEFILE ;
   (TRACEFILE (CAR l)))

(DF UNTRACE (%F ;; %X)
   ; enleve la TRACE des fonctions contenues dans %F ;
   ; ou bien des fonctions des precedents TRACE ;
   (OR
      %F
      (SETQ %F (AND (BOUNDP 'TRACE) TRACE))
      (LESCAPE "** UNTRACE quoi ?"))
   (MAPC %F
      (LAMBDA (%A)
         (COND
            ((MEMQ (TYPEFN %A) '(SUBR FSUBR))
               (STATUS 29 %A))
            ((AND
                (SETQ
                   %X (GETL %A '(EXPR FEXPR MACRO MACIN MACOUT)))
                (EQ (CAAR (CDDDR (CADR %X))) 'PRINTRACE))
               (REMPROP %A 'TRACE)
               (PUT %A
                  [LAMBDA
                   (CADR (CADR %X))
                   . (CDAR (CDAR (LAST (CADR (CDDDR (CADR %X))))))]
                  (CAR %X)))
            (T (LESCAPE (PRINT "** UNTRACE : C'est quoi " %A))))
         (SETQ TRACE (DELQ %A TRACE)) ))
   %F)
```

```
; STEPALL STEP UNSTEP ;

(DF STEPALL (%F)
   ; positionne le mode STEP ;
   ; (STEPALL T) le met ;
   ; (STEPALL)   l'enleve ;
   (STATUS (IF %F 1 2) 3 8) %F)

(DF STEP (%F ;; %X)
   ;;
   ; Force un STEP de n'importe quelle fonction de type ;
   ; EXPR FEXPR MACRO MACIN MACOUT ;
   ;;
   (OR
      (LISTP %F)
      ; y a rien a tracer ?!? ;
      (LESCAPE "** STEP quoi ?"))
   ; STEP contient la liste des fonctions (cf UNSTEP) ;
   ; pose des indicateurs ;
   (MAPC %F
      (LAMBDA (%A)
         (COND
            ((SETQ
                %X (GETL %A '(EXPR FEXPR MACRO MACIN MACOUT)))
               (PUT %A 0 'STEP)
               (PUT %A
                  [LAMBDA
                   (CADR (CADR %X))
                   ['PUT
                     [QUOTE %A]
                     ['ADD1 ['GET [QUOTE %A] ''STEP]]
                     ''STEP]
                   ['PRINTRACE
                     ['GET [QUOTE %A] ''STEP]
                     [QUOTE '--->]
                     [QUOTE %A]
                     [QUOTE ':]
                     (IF (ATOM (CADR (CADR %X)))
                        (CADR (CADR %X))
                        ['LIST . (CADR (CADR %X))])]
                   ['PRINTRACE
                     ['GET [QUOTE %A] ''STEP]
                     [QUOTE '<---]
                     [QUOTE %A]
                     [QUOTE '=]
                     ['PROG2
                        '(STATUS 1 8)
                        ['ETRACE [QUOTE  (CDDR (CADR %X))]]
                        '(STATUS 2 8)
                        ['PUT
                           [QUOTE %A]
                           ['SUB1 ['GET [QUOTE %A] ''STEP]]
                           ''STEP] ]]]
                  (CAR %X)))
            (T (SETQ %F (DELQ %F %A))
               (LESCAPE (PRINT "** STEP : C'est quoi " %A))))
         ; ajoute dans la liste des fonctions steppees ;
         (IF (BOUNDP 'STEP)
            (NEWL STEP %A)
            (SETQ STEP [%A]))))
   %F)
```

```
(DF UNSTEP (%F ;; %X)
   ; enleve le STEP des fonctions contenues dans %F ;
   ; ou bien des fonctions des precedents STEP ;
   (OR
      %F
      (SETQ %F (AND (BOUNDP 'STEP) STEP))
      (LESCAPE "** UNSTEP quoi ?"))
   (MAPC %F
      (LAMBDA (%A)
         (COND
            ((MEMQ (TYPEFN %A) '(SUBR FSUBR))
               (STATUS 29 %A))
            ((AND
                (SETQ
                   %X (GETL %A '(EXPR FEXPR MACRO MACIN MACOUT)))
                (EQ (CAAR (CDDDR (CADR %X))) 'PRINTRACE))
              (REMPROP %A 'STEP)
              (PUT %A
                 [LAMBDA
                  (CADR (CADR %X))
                  . (CDAR (CDAR (LAST (CADR (CDDDR (CADR %X))))))]
                 (CAR %X)))
            (T (LESCAPE (PRINT "** UNSTEP : C'est quoi " %A))))
         (SETQ STEP (DELQ %A STEP)) ))
   %F)
```

```
; COUNT COUNTFILE COUNTF ;

; COUNT et UNCOUNT permettent d'obtenir un comptage        ;
; du nombre d'appel dynamiques de differentes fonctions.   ;
; ce qui est tres utile pour determiner quelles fonctions  ;
; doivent etre ameliorer en priorite.                      ;


(DF COUNT (L ;; X)
   ; prepare le comptage du nombre d'appel ;
   ; des fonctions nommees dans la liste l ;
   ; ex. d'appel (COUNT FOO FUU FII)       ;
   (MAPC L
      (LAMBDA (L ;; X)
         (COND
            ((SETQ X (GETL L '(EXPR FEXPR MACRO MACIN MACOUT)))
               (ATTACH
                ['PUT
                 [QUOTE L]
                 ['ADD1 ['GET [QUOTE L] ''COUNT]]
                 ''COUNT]
                (CDDR (CADR X)))
               (PUT L 0 'COUNT))
            (T (PRINT "** COUNT : C'est quoi " L)))))
   L)


(DE COUNTFILE (file ;; l)
   ; lit et COUNT toutes les definitions du fichier file ;
   (INPUT file)
   (WITH (EOF () (INPUT) (&eof))
   (ESCLOOP &eof
      (SETQ l (READ))
      (EVAL l)
      (IF (AND (LISTP l)
               (MEMQ (CAR l) '(DE DF DM DMI DMO DEFUN)))
          (EVAL ['COUNT (CADR l)])))
   file))


(DF COUNTF (l)
   ; equivalent FSUBR de UNCOUNTFILE ;
   (COUNTFILE (CAR l)))
```

```
; SORTL UNCOUNT ;

(DE SORTL (L ;; RESL N Q L2 LL)
  ; trie la liste d'atomes L , une version de QUICKSORT ;
  ; Patrick GREU. Aout 78 ;
  (IF (NULL L) ()
      (SETQ RESL [NIL])
      (PUSH (LENGTH L) L)
      (LET ((L1))      ; L1 <doit> etre locale, voyez-vous pourquoi ? ;
         (SETQ L1 (POP) N (POP))
         (IF (EQN N 1) (LESCAPE L1))
         (SETQ Q (LOGSHIFT N -1)) (PUSH Q L1)
         (SETQ L1 (NTH Q L1))
         (PUSH (PLUS (LOGAND N 1) Q) (CDR L1))
         (RPLACD L1 NIL)
         (SETQ L1 (SELF) L2 (SELF) LL (RPLACD RESL L1))
         (WHILE L1
           (IF (SORT (CAR L2) (CAR L1))
               (SETQ N L1 L1 L2 L2 N N (RPLACD LL L1)))
           (SETQ LL L1 L1 (CDR L1)))
         (RPLACD LL L2) (CDR RESL))))

(DF UNCOUNT (L)
   ; edite le nombre de fois que les fonctions                ;
   ; precedement mentionnees par COUNT ont ete appellees      ;
   ; UNCOUNT enleve le comptage des fonctions                 ;
   ; ex. d'appel : (UNCOUNT)                                  ;
   (SETQ L
    (SORTL (OR L
               (MAPCT (OBLIST) (LAMBDA (L) (AND (GET L 'COUNT) L))))))
   (MAPC L
      (LAMBDA (L ;; N)
         (COND
            ((SETQ N (GET L 'COUNT))
               (PRIN1 L ':)
               (TTAB 15)
               (PRINT N)
               (REMPROP L 'COUNT)
               (SMASH (CDDR (OR (GET L EXPR) (GET L FEXPR)))))
            (T (PRINT "** UNCOUNT : C'est quoi " L)))))
   L)

; epilogue standard ;

(READ.TABLE READ.TABLE)

(POUR EVAL
   (MAPC (MAPCAR (MAKLIST "SYS:DEBUG.VLI loaded.") 'CASCII)
      'TYO)
   (TERPRI)
   'DEBUG)
```

APPENDICE D

```
;        P R E T T Y   C R O S S - R E F E R E N C E          ;
;                                                              ;
;             Editeur / Indexeur   VLISP 10 . 3                ;
;--------------------------------------------------------------;
;                  Jerome CHAILLOUX                            ;
;                                                              ;
;        Departement d'Informatique                            ;
;        Universite de Paris VIII - Vincennes                  ;
;        Route de la Tourelle 75571 Paris Cedex 12             ;
;        Tel : 374 12 50 poste 299                             ;
;                                                              ;
;        I.R.C.A.M.                                            ;
;        31 Rue St Merri 75004 Paris                           ;
;        Tel : 277 12 33 poste 48-48                           ;
;--------------------------------------------------------------;
; Pour pouvoir lire ce fichier en tous temps ... ;

(PROGN
   ; sauvetage de la table du READ ;
   (SETQ READ.TABLE (READ.TABLE))
   ; passage sur la table standard ;
   (READ.STD)
   ; voila c'est pret ;
   'PRET)
```

```
; 3 %TAB ADDN FLATSIZE SARENTRE SORTL ;

(MAPC '(/( /) /. /  /') (LAMBDA (X) (SET X X)))

(OR (BOUNDP '?index) (SETQ ?index NIL))

(DE %TAB (col) (STATUS 7 (PLUS col (STATUS 7))))

(DE ADDN (X) (AND (LT %T (SETQ %N X)) (EXIT)) T)

(DE FLATSIZE (%L ;; %N)
   ; calcule la taille de l'expression %L ;
   (COND
      ((ATOM %L) (PLENGTH %L))
      ((AND (EQ (CAR %L) QUOTE) (NULL (CDDR %L)))
         (ADD1 (FLATSIZE (CADR %L))))
      ((EQ (CAR %L) 'LIST) (FLATSIZE (CDR %L)))
      ((EQ (CAR %L) 'COMMENT) (PLUS 2 (LENGTH (CDR %L))))
      ((ADDN (PLUS 2 (FLATSIZE (NEXTL %L))))
         (WHILE (LISTP %L)
            (ADDN (PLUS %N (ADD1 (FLATSIZE (NEXTL %L))))))
         (AND %L (ADDN (PLUS %N 3 (FLATSIZE %L))))
         %N)))

(DE SARENTRE (%E ;; %T)
   ; predicat qui teste si %E rentre dans la ligne courante ;
   (ESCAPE EXIT
      (SETQ %T (DIFFER (STATUS 9) (STATUS 8) 5))
      (GT %T (FLATSIZE %E))))

(DE SORTL (L ;; RESL N Q L2 LL)
   ; trie la liste d'atomes L , une version de QUICKSORT ;
   ; Patrick GREUSSAY. Aout 78 ;
   ; FFFFFFFFFFRRRRRRRRRRRR........ ;
   (IF (NULL L)
      NIL
      (SETQ RESL [NIL])
      (PUSH (LENGTH L) L)
      (LET ((L1))
         ; L1 <doit> etre locale, voyez-vous pourquoi ? ;
         (SETQ L1 (POP) N (POP))
         (IF (EQN N 1) (LESCAPE L1))
         (SETQ Q (LOGSHIFT N -1))
         (PUSH Q L1)
         (SETQ L1 (NTH Q L1))
         (PUSH (PLUS (LOGAND N 1) Q) (CDR L1))
         (RPLACD L1 NIL)
         (SETQ L1 (SELF) L2 (SELF) LL (RPLACD RESL L1))
         (WHILE L1
            (IF (SORT (CAR L2) (CAR L1))
               (SETQ N L1 L1 L2 L2 N N (RPLACD LL L1)))
            (SETQ LL L1 L1 (CDR L1)))
         (RPLACD LL L2)
         (CDR RESL))))
```

```
; 4 %EOL %TERPRI %PC XPRDOT XSUIV %PPTC %P1 ;

(DE %EOL ()
   ; TESTE SI Y FO UN TERPRI ;
   (IF (GT (STATUS 8) (STATUS 7)) (%TERPRI) (STATUS 8 (STATUS 7))))

(DE %TERPRI (%N)
   ; effectue un TERPRI en numerotant les lignes ;
   (TERPRI %N)
   (IF ?index
      (PROGN (TTAB 0) (PRIN1 (INCR #nbline)) (TTAB (STATUS 7)))))

(DF %PC (%L)
   ; imprime un commentaire ;
   (%TERPRI)
   (PRINC PTVRG 1)
   (EPROGN %L)
   (SPACES 1)
   (PRINC PTVRG 1)
   (%TERPRI 2))

(DE %PRIN1 (%L)
   ; simule PRIN1 et fabrique l'index ;
   (PRIN1 %L)
   (IFN ?index (LESCAPE %L))
   (IF (AND %L (LITATOM %L) (OR ?crossall (GT (LOC %L) (LOC 'STOP))))
      (PUT %L [#nbline . (GET %L 'CROSS)] 'CROSS))
  %L)

(DE XPRDOT () (PRINC /.) (SPACES 1) (%PRIN1 %E))

(DE XSUIV ()
   (AND
      (ATOM %E)
      (EXIT (PRINC /. 1) (SPACES 1) (%PRIN1 %E) (PRINC /) 1)))
   (NEXTL %E))

(DE %PPTC ()
   ; edite sur la meme ligne le commentaire (s'il existe) ;
   ; en tete de %E et met a jour %E ;
   (IF (EQ (CAR %E) 'COMMENT) (PROGN (%P1) (NEXTL %E))))

(DE %P1 ()
   ; imprime sur la meme ligne l'objet suivant de %E ;
   (%PPTC)
   (COND
      ((SARENTRE (CAR %E))
         (SPACES 1)
         (COND
            ((CAR %E) (SUPERP (NEXTL %E)))
            ((PRINC /( 1) (PRINC /) 1) (NEXTL %E))))
      (T (%TAB 1) (SPACES 1) (%EOL) (SUPERP (NEXTL %E)) (%TAB -1)))
   (%PPTC))
```

```
; 5 %PP XPRTTY XPRTTY1 XPRTTY2 ;

(DE %PP (%E)
   ; imprime un : PROG DO LAP LOAD ;
   (%TAB 7)
   (MAPC %E
      (LAMBDA (%E)
         (COND
            ((ATOM %E)
               (%TERPRI)
               (STATUS 8 (DIFFER (STATUS 7) 7))
               (%PRIN1 %E)
               (%EOL))
            ((EQ (CAR %E) 'COMMENT)
               (IF (SARENTRE %E) (SPACES 1) (%TERPRI))
               (SUPERP %E))
            (T (%EOL) (SUPERP %E)))))
   (PRINC /))
   (%TAB -7))

(DE XPRTTY ()
   ; edite les differents elements de %E  (normalement) ;
   (COND
      ((SARENTRE %E) (WHILE %E (SPACES 1) (SUPERP (XSUIV %E))))
      ((SPACES 1)
         (%TAB 3)
         (MAPC %E (LAMBDA (%E) (%EOL) (SUPERP %E)))
         (%TAB -3)))
   (PRINC '/) 1))

(DE XPRTTY1 ()
   ; edite 1 element sur la meme ligne ;
   ; et le reste normalement ;
   (%P1)
   (XPRTTY))

(DE XPRTTY2 ()
   ; edite des clauses de 'COND' ou de 'SELECT..' ;
   (%TAB 3)
   (MAPC %E
      (LAMBDA (%E)
         (%EOL)
         (IF (EQ (CAR %E) 'COMMENT)
            (LESCAPE (SUPERP %E))
            (PRINC /( 1)
            (%TAB 1)
            (SUPERP (NEXTL %E))
            (%TAB -1)
            (XPRTTY))))
   (%TAB -3)
   (PRINC /) 1))
```

```
; 6 SUPERP ;

(DE SUPERP (%E ;; %L %F %T)
   ; edite %E ;
   (IF (ATOM %E) (LESCAPE (%PRIN1 %E)))
   ; c'est donc une liste ;
   ; %F = le type de la fonction a editer ;
   (SETQ
      %F (OR
         (AND (LITATOM (CAR %E)) (GET (CAR %E) 'PRETTY))
         (CAR %E)))
   (ESCAPE EXIT
      (COND
         ((AND (EQ %F QUOTE) (NULL (CDDR %E)))
            (SETQ %E (CADR %E))
            (AND (ATOM %E) (NOT (SARENTRE %E)) (%TERPRI))
            (PRINC /')
            (SUPERP %E))
         ((AND (LISTP (CAR %E)) (EQ (CAAR %E) LAMBDA))
            (SETQ %L (CADAR %E) %T (CDR %E))
            (SUPERP
             ['LET
              (MAPCAR %L (LAMBDA (X) [X (NEXTL %T)]))
             . (CDDAR %E)]))
         ((AND (EQ %F 'MCONS) (NULL (CDR (LAST %E))))
            ; (MCONS x1 ... xN)  [ x1 ... xN-1 . xN] ;
            (PRINC '/[)
            (NEXTL %E)
            (COND
               ((SARENTRE %E)
                  (WHILE (LISTP (CDR %E))
                     (SUPERP (NEXTL %E))
                     (SPACES 1)))
               (T (%TAB 1)
                  (WHILE (LISTP (CDR %E))
                     (SUPERP (NEXTL %E))
                     (%TERPRI))
                  (%TAB -1)))
            (PRINC /.)
            (PRINC / )
            (SUPERP (CAR %E))
            (PRINC '/]))
         ((AND
             (EQ %F 'CONS)
             (NULL (CDR (LAST %E)))
             (NULL (CDDDR %E)))
            ; (CONS x1 x2)  [x1 . x2] ;
            (SUPERP
             ['MCONS . (IF (CDDR %E) (CDR %E) [(CADR %E) NIL])]))
         ((AND (EQ %F 'NCONS) (NULL (CDDR %E)))
            ; (NCONS x1)   [x1] ;
            (SUPERP ['LIST . (CDR %E)]))
```

```
; 7 SUPERP (suite) ;
;;
((AND (EQ %F 'LIST) (NULL (CDR (LAST %E))))
   ; (LIST x1 ... xN)   [x1 ... xN] ;
   (PRINC '/[)
   (NEXTL %E)
   (COND
      ((SARENTRE %E)
         (WHILE (LISTP %E)
            (SUPERP (NEXTL %E))
            (AND %E (SPACES 1))))
      (T (%TAB 1)
         (WHILE (LISTP %E)
            (SUPERP (NEXTL %E))
            (AND %E (%TERPRI)))
         (%TAB -1)))
   (AND %E (XPRDOT))
   (PRINC '/]))
((AND (EQ %F 'FUNCTION) (NULL (CDDR %E)))
   (PRINC /()
   (%PRIN1 (NEXTL %E))
   (SPACES 1)
   (SUPERP (CAR %E))
   (PRINC /)))
((EQ %F 'COMMENT)
   (AND (EQUAL (CDR %E) '(P A G E)) (EXIT (PAGE)))
   (PRINC PTVRG)
   (%TAB (SETQ %T (DIFFER (STATUS 8) (STATUS 7))))
   (MAPC (CDR %E) 'PRINC)
   (%TAB (MINUS %T))
   (PRINC PTVRG))
((EVERY %E 'ATOM)
   (PRINC /()
   (%TAB (SETQ %T (DIFFER (STATUS 8) (STATUS 7))))
   (WHILE (LISTP %E)
      (OR (SARENTRE (CAR %E)) (%TERPRI))
      (%PRIN1 (NEXTL %E))
      (AND %E (SPACES 1)))
   (AND %E (XPRDOT))
   (PRINC /))
   (%TAB (MINUS %T)))
```

```
; 8 SUPERP (suite) ;
;;
((PRINC /()
   (WHILE %E
      (SETQ %L (XSUIV %E))
      (COND
         ((NULL %E))
         ((OR
             (MEMQ %L %S)
             (MEMQ %F
              '(OR AND LIST MCONS NCONC1 APPEND1 PLUS DIFFER
                TIMES QUO MIN MAX PUSH EXPLODE GENSYM
                PATHLIBRARY LESCAPE PROGN PROG1 PROG2)))
            (%PRIN1 %L)
            (EXIT (XPRTTY)))
         ((MEMQ %F '(SET SETQ SETQQ))
            (%PRIN1 %L)
            (COND
               ((SARENTRE %E)
                  (WHILE %E (SPACES 1) (SUPERP (NEXTL %E))))
               ((%TAB 3)
                  (WHILE %E
                     (%TERPRI)
                     (SUPERP (NEXTL %E))
                     (IF %E
                        (%P1)
                        (EXIT (PRINC /)) (%TAB -3))))
                  (%TAB -3)))
            (EXIT (PRINC '/))))
         ((MEMQ %L '(ESCAPE ESCLOOP))
            (%PRIN1 %L)
            (SETQ %S [(CAR %E) . %S])
            (EXIT (XPRTTY1) (NEXTL %S)))
         ((MEMQ %F
           '(DE DF DM DMI DMO DMC DG MACLAP MACMP DGEN))
            ; tous les types de definitions standards ;
            (%PRIN1 %L)
            (SPACES 1)
            (IF (LITATOM (CAR %E))
               ; forme normale des DEFs ;
               (PROGN (SUPERP (NEXTL %E)) (EXIT (XPRTTY1)))
               ; call-forme des DEFs ;
               (SUPERP (NEXTL %E))
               (EXIT (XPRTTY))))
         ((MEMQ %F
           '(LAMBDA WITH WHILE UNTIL STATUS BOOLE PUT IFN
             GAMMA IF EVERY SOME ORF ANDF REPEAT REPEATUNTIL
             MAP MAPC MAPS MAPLIST MAPSUB MAPT MAPCT MAPST
             POUR LET)) (%PRIN1 %L) (EXIT (XPRTTY1)))
         ((EQ %F 'PROG)
            (%PRIN1 %L)
            (SPACES 1)
            (SUPERP (NEXTL %E))
            (EXIT (%PP %E)))
         ((EQ %F 'LAP)
            (%PRIN1 %L)
            (OR (EQ (CAAR %E) QUOTE) (EXIT (XPRTTY)))
            (SPACES 1)
            (PRINC /')
            (PRINC /()
            (%PP (CADAR %E))
            (NEXTL %E)
            (EXIT (XPRTTY)))
```

```
                    ((MEMQ %F '(BACKTRACK SELECT SELECTQ))
                       (%PRIN1 %L)
                       (SPACES 1)
                       (%P1)
                       (EXIT (XPRTTY2)))
                    ((EQ %F 'COND) (%PRIN1 %L) (EXIT (XPRTTY2))))
                 (%TAB 1)
                 (OR (SARENTRE %L) (%EOL))
                 (SUPERP %L)
                 (AND %E (SPACES 1))
                 (%TAB -1))
              (PRINC '/))))))
```

```
; 9 PRETTYNIT PRETTYFIN PRETTYP PRETTY ;

(DE PRETTYNIT ()
   ; initialisation des STATUS avant edition ;
   (PUSH (STATUS 7) (STATUS 9) (STATUS 0))
   (STATUS 7 (IF ?index 6 1))
   (STATUS 9 PRETTYSIZE)
   (STATUS 2 19 25)
   (STATUS 1 24 26 27))

(DE PRETTYFIN ()
   ; restauration des STATUS apres edition ;
   (MAPC '(0 9 7) (LAMBDA (%X) (STATUS %X (POP))))
   (TERPRI))

(DE PRETTYP (%E ;; %S)
   ; PRETTY-PRINT une S-expression ;
   ; %S est la table des ESCAPEs actifs ;
   (PRETTYNIT)
   (SUPERP %E)
   (PRETTYFIN)
   %E)

(DF PRETTY (%L ;; %X)
   ; PRETTY-PRINT des fonctions ;
   (MAPC %L
      (LAMBDA (%L)
         (%TERPRI)
         (COND
            ((SETQ
               %X
                (GETL %L
                 '(EXPR FEXPR MACRO MACIN MACOUT MACLAP !macmp !gen)))
              ; cas des fonctions normales ;
              (PRETTYP
               [(CASSQ (CAR %X)
                 '((EXPR . DE) (FEXPR . DF) (MACRO . DM)
                  (MACIN . DMI) (MACOUT . DMO) (MACLAP . MACLAP)
                  (!macmp . MACMP) (!gen . DGEN)))
                %L
                . (CDR (CADR %X))]))
            ((AND (EQ (PLENGTH %L) 1) (SETQ %X (STATUS 18 %L)))
              ; cas des macros-caracteres ;
              (PRETTYP ['DMC %L . (CDR %X)]))
            ((BOUNDP %L) ; on essaie la C-val ; (PRETTYP (CAR %L)))
            (T ; on sait pas ce que c'est ;
              (PRINT "** PRETTY : C'est quoi ?" %L)))
         (%TERPRI))))
```

```
; 10 PRETTYFILE ;

(DE PRETTYFILE (filout filin ?swl ;; %S lu lastprtcom)
   ; ?swl = T si on laisse les casses en l'etat ;
   ; %S est la liste des ESCAPEs actifs ;
   ; lastprtcom sert a controler la supression de lignes ;
   ; blanches entre 2 commentaires ;
   (DE EOF ()
      (REMPROP 'TOPLEVEL EXPR)
      (REMPROP 'EOF EXPR)
      (PRETTYFIN)
      (IFN ?index NIL (STATUS 7 0) (PAGE) (CROSSPRINT))
      (%PC
       (PRIN1 (IF (BOUNDP 'VERSION) VERSION (VERSION)) (DATE) (TIME)
        'END 'OF 'FILE ': filin))
      (STATUS 17 (ASCII 13) 0)
      (STATUS 19 PTVRG)
      (STATUS 19 (ASCII 12))
      (STATUS 1 18 19)
      (OUTPUT)
      (INPUT)
      (&EOF filout))
   (DE TOPLEVEL ()
      (SETQ lu (READ))
      (COND
         ((ATOM lu) (PRINT lu))
         ((AND (EQ (CAR lu) 'POUR) (EQ (CADR lu) 'PRETTY))
            (EPROGN (CDDR lu)))
         ((NEQ (CAR lu) 'COMMENT)
            (AND lastprtcom (%TERPRI))
            (IF
             (AND ?index (MEMQ (CAR lu) '(DE DF DM DMI DMO DMC DA)))
               (SETQ
                  #nbfnt [[(CADR lu) (CAR lu) #nbline] . #nbfnt]))
            (SUPERP lu)
            (SETQ lastprtcom)
            (%TERPRI))
         (T (SETQ lastprtcom T) (SUPERP lu)))
      (%TERPRI))
   (STATUS 17 (ASCII 13) 1)
   (STATUS 18 (ASCII 12) (LAMBDA () ['COMMENT 'P 'A 'G 'E]))
   (SETQ PTVRG (ASCII 59))
   (STATUS 18
      PTVRG
      (LAMBDA (%L %C %T)
         (SETQ %L ['COMMENT])
         (UNTIL (EQ (SETQ %C (READCH)) PTVRG) (SETQ %L [%C . %L]))
         (REVERSE %L)))
   (OUTPUT (OR filout '(DSK (PRETTY . VLP) (GETPPN) 45)))
   (%PC
    (PRIN1 (IF (BOUNDP 'VERSION) VERSION (VERSION)) (DATE) (TIME)
     'FILE ': filin))
   (PRETTYNIT)
   (STATUS (IF ?swl 2 1) 19)
   (STATUS 2 18)
   (INPUT filin)
   (ESCLOOP &EOF (TOPLEVEL)))
```

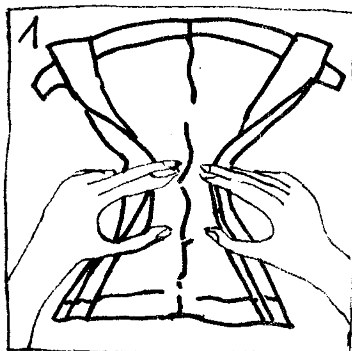
```
; 11 PRETTYF PRETTYSIZE PRETTYEND ;

(DF PRETTYF (filin)
   ; PRETTY-PRINT un fichier standard ;
   ; le 1er arg est le nom du fichier, le 2eme l'indic ;
   ; ex : (PRETTYF PRETTY) , (PRETTYF LODLAP T) ;
   (PRETTYFILE ['DSK [(CAR filin) . 'VLP] (GETPPN) 45]
    ['DSK [(CAR filin) . 'VLI]] (CADR filin)))

(DE PRETTYSIZE (%N)
   ; initialise la largeur d'impression du PRETTY ;
   (SETQ PRETTYSIZE (IF (NUMBP %N) %N 71)))

(DE PRETTYEND (;; X)
   ; recupere la place de PRETTY-PRINT ;
   (SETQ X (STATUS 21))
   (MAPC
    '(%TAB ADDN FLATSIZE SARENTRE %EOL XPRDOT XSUIV %P1 %PP XPRTTY
      XPRTTY1 XPRTTY2 SUPERP PRETTYP PRETTYFILE PRETTYSIZE PRETTYEND
      CROSSPRINT CROSS CROSSFILE) '(LAMBDA (X) (REMPROP X 'EXPR)))
   (MAPC '(PRETTY PRETTYF CROSSF) (LAMBDA (X) (REMPROP X 'FEXPR)))
   (AUTOLOAD PRETTY PRETTYP PRETTY PRETTYFILE PRETTYF PRETTYSIZE
    PRETTYEND)
   (PRINT 'PRETTYEND '= (DIFFER (STATUS 21) X)))

(OR (BOUNDP 'PRETTYSIZE) (SETQ PRETTYSIZE 71))
```



```
; 12 CROSSPRINT ;

(DE CROSSPRINT (;; #nb X)
   ; edite le resultat du CROSS ;
   ; les atomes references possedent sous l'indicateur ;
   ; 'CROSS la liste des numeros de lignes. ;
   (SETQ ?index NIL)
   (SETQ #nb 0)
   (STATUS 2 20)
   (TTAB 17)
   (PRINT 'CROSS 'REFERENCE)
   (TERPRI 2)
   ; impression de la liste des fonctions ;
   (MAPC (SORTL (MAPCAR #nbfnt 'CAR))
      (LAMBDA (L)
         (MAPC
          (LET ((X #nbfnt))
             (COND
                ((NULL X) NIL)
                ((EQ (CAAR X) L) (CAR X))
                (T (SELF (CDR X))))) 'PRIN1)
         (TERPRI)))
   (TERPRI 2)
   ; edition du cross-ref ;
   (MAPC (SORTL (MAPCT (OBLIST) (LAMBDA (L) (AND (GET L 'CROSS) L))))
      (LAMBDA (L)
         (SETQ X (GET L 'CROSS))
         (REMPROP L 'CROSS)
         (STATUS 7 0)
         (TERPRI)
         (PRIN1 (INCR #nb))
         (TTAB 4)
         (PRIN1 L)
         (STATUS 7 18)
         (TTAB 17)
         (MAPC (FREVERSE X) (LAMBDA (X) (PRIN1 X)))
         (TERPRI)))
   (STATUS 7 0)
   (TERPRI))
```

```
; 13 CROSS CROSSFILE CROSSF ;

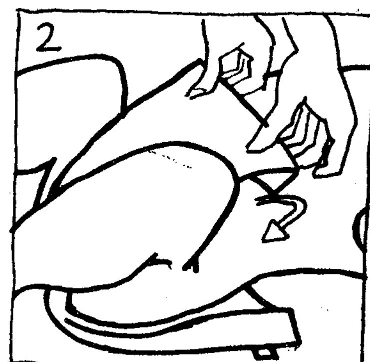
(DF CROSS (F ;; ?crossall ?index #nbline #nbfnt)
   (SETQ #nbline 0)
   (SETQ ?index T)
   (SETQ ?crossall (CADR F))
   (STATUS 7 9)
   (EVAL ['PRETTY (CAR F)])
   (CROSSPRINT))

(DE CROSSFILE
 (filout filin ?crossall ?crossbdc ;; ?index #nbline #nbfnt)
   ; CROSS-REF 'filin' dans 'filout' ;
   ; ?crossall = T si on veut tous les atomes systemes ;
   ; ?crossbdc = T si on veut les bas-de-casses ;
   (SETQ #nbline 0)
   (SETQ ?index T)
   (STATUS 7 9)
   (SETQ PRETTYSIZE (PLUS PRETTYSIZE 10))
   (OR filout (SETQ filout '(DSK (CROSS . VLC) (GETPPN) 45)))
   (PRETTYFILE filout filin ?crossbdc)
   (SETQ PRETTYSIZE (DIFFER PRETTYSIZE 10))
   filout)

(DF CROSSF (filin)
   ; CROSS-REF un fichier standard ;
   ; appel : (CROSSF nom de la file , ?crossall , ?crossbdc) ;
   ; ex : (CROSSF PRETTY T T) ;
   (CROSSFILE ['DSK [(CAR filin) . 'VLC] (GETPPN) 45]
    ['DSK [(CAR filin) . 'VLI]] (CADR filin) (CADDR filin)))

(READ.TABLE READ.TABLE)

(POUR EVAL
   (MAPC (MAPCAR (MAKLIST "SYS:PRETTY.VLI loaded.") 'CASCII) 'TYO)
   (TERPRI)
   'PRETTY)
```

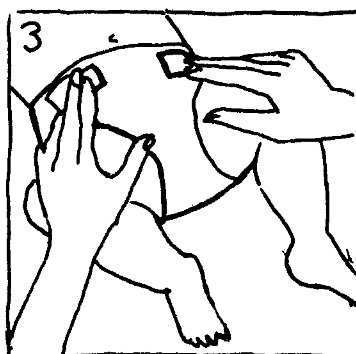


CROSS REFERENCE

%EOL DE 88
%P1 DE 128
%PC DF 98
%PP DE 143
%PPTC DE 123
%PRIN1 DE 107
%TAB DE 36
%TERPRI DE 92
ADDN DE 38
CROSS DF 544
CROSSF DF 566
CROSSFILE DE 552
CROSSPRINT DE 503
FLATSIZE DE 40
PRETTY DF 387
PRETTYEND DE 486
PRETTYF DF 475
PRETTYFILE DE 416
PRETTYFIN DE 374
PRETTYNIT DE 366
PRETTYP DE 379
PRETTYSIZE DE 482
SARENTRE DE 54
SORTL DE 60
SUPERP DE 196
XPRDOT DE 115
XPRTTY DE 161
XPRTTY1 DE 171
XPRTTY2 DE 177
XSUIV DE 117

1		32 233
2	!gen	396 402
3	!macmp	396 402
4	#nb	503 508 531
5	#nbfnt	447 447 514 517 544 553
6	#nbline	96 112 447 544 545 553 557
7	%C	458 460 460
8	%E	54 58 115 119 120 121 126 126 132 135 135 136 137 143 146 147 149 152 154 155 156 157 164 164 164 167 167 167 180 181 183 184 187 196 198 198 203 203 204 207 208 208 209 209 211 212 212 213 213 217 218 221 223 224 225 228 229 234 238 239 242 242 242 243 245 248 251 253 254 255 256 258 259 260 262 264 266 268 271 274 277 280 281 282 283 284 290 291 293 305 306 306 308 310 311 318 325 327 329 339 340 343 347 348 359 379 383 385
9	%EOL	88 137 153 157 167 182 357 490
10	%F	196 202 207 218 237 243 248 264 270 296 302 320 331 336 341 350 355

70	SUPERP	135 137 156 157 164 167 184 187 196 211 214 225 229 234 241 245 255 259 268 306 310 327 329 339 358 383 448 451 491
71	VLC	561 570
72	VLI	480 571
73	VLP	462 479
74	WITH	332
75	X	32 32 32 38 38 216 216 486 488 492 492 493 493 496 503 517 519 520 520 521 527 536 536 536
76	XPRDOT	115 262 284 490
77	XPRTTY	161 175 189 301 330 343 349 490
78	XPRTTY1	171 319 327 335 491
79	XPRTTY2	177 354 355 491
80	XSUIV	117 164 291 490
81	[	220 250
82	]	235 263
83	col	36 36
84	filin	416 428 465 469 475 479 480 480 553 562 566 570 571 571 571
85	filout	416 435 462 553 561 561 562 564
86	lastprtcom	416 443 449 451
87	lu	416 437 439 439 440 440 441 442 445 447 447 448 451



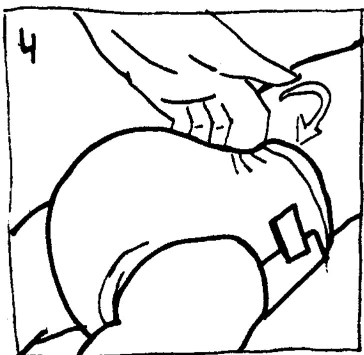

4

TABLE D'INDEX du MANUEL VLISP 10 . 3

